(2)

知識ベースビューに基づく
エキスパートシステム構築に関する研究

平成５年２月

辻　　洋

目　　次

第１章　序　　論

１．１　本研究の背景

近年、エキスパートシステム構築ツールの普及とともに、業務専門家（以下エキスパートと呼ぶ）がもつ知識を計算機に蓄積し利用するエキスパートシステムの開発が数多く報告されている[Sasaki, et al. 1986][Hirai. 1987]。一般にエキスパートシステムは、ナレッジエンジニアと呼ばれるエキスパートシステム構築ツールに精通した計算機専門家により構築される。ナレッジエンジニアは知識を収集するため、エキスパートにインタビューしたり、エキスパートの問題解決事例を分析する。

しかし、次の理由によりエキスパートシステムの構築は必ずしも容易ではない。

① 現在普及している大半のエキスパートシステム構築ツールは知識表現としてルールやフレームを採用している。これらの知識表現法は応用領域に関して汎用的であるが、その抽象レベルとエキスパートのもつ問題解決に関する知識の抽象レベルが異なるため、収集した知識の表現法は一意に決まらない。

②「エキスパートは、自分が思っている以上に知っている」といわれるようにエキスパートシステム開発時に、正確かつ完全な知識を収集することは困難である。場合によっては、環境の変化により知識が陳腐化したり、新しい知識が得られることもある。

そのため、種々のエキスパートシステム構築手法と知識獲得ツールが研究されている。

上記①の問題に対して、抽象度がエキスパートに近い問題解決手法(generic task [Chandrasekaran. 1985]と呼ばれる）やそれに基づくエキスパート向け知識獲得ツール（MOLE[Eshelman, et al. 1986], CSRL[Bylander, et al. 1986]など）が研究されている。これらは、診断や分類など応用分野を制約した上でエキスパートの問題解決手法を個別にモデル化し、そのモデルに基づいて、ツールが直接エキスパートにインタビューすることにより知識を獲得しようという試みである。しかし、一口に診断あるいは分類といっても種々の問題解決手法があり、いくつの generic task があるのか明確でない現在、知識獲得のインタビュー機能を有するツールを網羅的に開発することは合理的ではない。

上記②の問題に対して、知識を逐次洗練していくアルゴリズム（SEEK2[Ginsberg, et al. 1985] など）や問題解決例から知識を学習するアルゴリズム（ID3[Quinlan. 1983] など）の研究が行われている。しかし、知識が不完全な時からエキスパートシステムが運用されることを前提とした問題解決モデルについてはあまり考察されていない。

つまり、従来の研究は知識表現、あるいは知識獲得など知識を計算機に移植するための技術課題に重きをおくあまり、誰がどうエキスパートシステムの構築に参加し、同一の知識ベースに対して、誰がどう見るかなどの運用面の課題について未解決であった。

１．２　本研究の目的

本論は、知識ベースビューという概念を考慮したエキスパートシステムの構築に係る一連の研究をとりまとめたものである。基本的なアイデアは、ナレッジエンジニアが開発した知識ベースに対し、知識ビューと呼ぶ一種のメタ知識を与えることにより、エキスパートやエンドユーザが彼らの抽象レベルで知識にアクセスできる手段を提供しようというものである。

この知識ベースビューは、データベース（ＤＢ）技術のビューという概念[Date. 1981][Ullman. 1980]の影響を受けている。ＤＢビューは、仮想的なデータ構造（利用者に対するデータベースの見え方）を定義する手段、エンドユーザが意識しなくてよい部分にマスクを与えデータを保護する手段、などを提供する。すなわち、ＤＢビューは利用者に対するヒューマンインタフェースを与えるものである。

知識ベースビューは、これらＤＢビューの概念を知識処理に発展させ、知識ベースの構築のためのヒューマンインタフェースを提供することを目的とする。

本論で述べる研究は以下の二つの段階からなる。

（１）エキスパートシステムの構築モデルに知識ベースビューの概念を導入し、システム開発、再利用のモデルを提案する。[Tsuji, et al. 1989b]

提案するモデルは、従来のツールが KEE[Fikes, et al. 1985] や ART のように汎用性を指向したナレッジエンジニア用と、 ETS[Boose. 1984] や MOLE [Eshelman, et al. 1986] のように専用性を指向したエキスパート用とに分けて論じられたために生じる問題を解決するものである。つまり、ナレッジエンジニアにより開発された知識ベースをエキスパートが保守する困難性、エキスパートがプロトタイピングした知識ベースをナレッジエンジニアが拡張する困難性を除去することを目的とする。

（２）具体的な応用例に基づいて、知識ベースビューの表現、機能、有効性、限界について考察する。

特に保守という観点から、①静的に（推論とは独立に）保守する場合[Tsuji, et al. 1988b]、②動的に（推論過程を通して）保守する場合[Tsuji, et al. 1988c]③リレーショナル・データベースの研究成果を用いた場合[Tsuji, et al 1989a]について個別に論じる。これらの応用例が、初期インストールとしてとりあえずの知識だけで動作し、知識ベースビューを用いて、順次エキスパートにより知識を更新されるように設計されることを示す。

本論では、はじめに、計算機部門による定形処理のプログラミング、エンドユーザ部門における非定形処理のプログラミングとを対比することにより、エキスパートシステムの関与者と構築手順を分析する。その結果、一つの知識ベースに対して次の４種類のインタフェースを提案する：

（ａ）試行環境：エキスパート自身がシステムのイメージ作りを行う。
（ｂ）開発環境：ナレッジエンジニアが本格的にエキスパートシステムを構築する。
（ｃ）利用環境：エンドユーザがエキスパートシステムを利用する
（ｄ）保守環境：エキスパートが知識ベースを洗練化していく。

この中では開発環境がエキスパートシステムを構築する上での基本となる機能を提供する環境と考え、開発環境の上に他の環境を統合するために知識ベースビューと呼ぶ概念を導入する。

試行環境は、過去にナレッジエンジニアによりシステム化された問題解決と同じ手法が使える場合、エキスパートが短時間で小さな知識ベースを作成できるようにする。エキスパートが、エキスパートシステムとは如何なるものであり、何ができるかを知るためのインタフェースである。そのため、試行環境では、どのような問題解決手法があるかが分かり、その一つを選択したとき、開発環境で参照するよりも抽象度の高い形式で知識を入力できるように知識ベースビューを導入する。

試行環境で取り揃えられる問題解決のメニューは、あらゆる問題を解決できるわけでなく、生成される知識ベースは必ずしも新しい分野で要求される機能を満足できない。そのときは、ナレッジエンジニアが開発環境で応用領域に関して汎用的な知識表現を用いて知識ベースを作成する。

利用環境を実現する知識ベースビューは、如何なる知識の集合（知識セット）で一つの応用システムが構成されるかを定義する。利用者は どの知識セット（あるいは 誰の知識セット）を用いて問題を解くかを選択したり、問題の解がどの知識セットに基づくかを知ることができる。一方、この知識ベースビューは、ある知識の集合（知識セット）が一つ以上のエキスパートシステムに含まれることも可能とする。

保守環境を実現する知識ベースビューは、エキスパートが 自分の責任範囲について自分の見やすい表現で知識ベースにアクセスすることを可能とする。知識の保守とは、単なる維持・管理だけでなく、内容の更新までも意味する。ある応用で保守のために定義された知識ベースビューは、問題解決手法が同じ別の応用を試行するときに利用できる。例えば、ある分類型のエキスパートシステムの保守ビューの組ができると、これを試行ビューとして登録することにより、以後 類似した分類型の応用システムのイメージ作成を行う試行環境ができる。

このように 知識ベースビューは エキスパートシステムのアーキテクチュアの再利用も可能とする。

次に、３種類の具体的な知識ベースビューについて考察する。

（１）知識ベースを静的に保守するための知識ベースビュー

この知識ベースビューは、新たな問題解決モデルが考案される都度生じていた個別に知識獲得ツールを開発する労力を省力化するものである。この知識ベースビューを実現するために、ナレッジエンジニアにより開発される知識ベースに 次の仮定をおく：

・エキスパートの知識により継続的に保守される部分（問題領域に関する知識）と　基本的に変更されない部分（推論制御の知識）がある。
・継続的に保守される部分においても陽に定義できる構造がある。

この仮定に基づき、知識ベースの構造と属性を表現するメタ知識とそのインタプリタを提案する。つまり、ナレッジエンジニアのための知識表現とエキスパートのための知識表現とを相互に変換する規則をメタ知識として定義することにより、エキスパートによる知識ベースの編集を可能とする広義のエディタを生成する。

生成されたエディタは、知識ベースビューとして エキスパートが 知識ベースの如何なる部分をアクセスできるか示し、フィルインザブランク形式でエキスパートの操作を誘導し、重複した知識や冗長な知識の警告を行う。

次に、提案した知識ベースビューをプログラミングノウハウを伝承するためのエキスパートシステム[Tsuji, et al. 1988d]に適用し、先の４つの環境がどのように構築されていくかを検証する。

このエキスパートシステムは、１６１件のソフトウエア事故に関する教訓的なノウハウ（１件１葉形式の文書）を蓄積しており、このエキスパートシステムの問題は、新たに生じた事故を既に知られている教訓に結び付けることである。

具体的な事故と格言化した教訓では その抽象度が違うので、単純なキーワード検索で両者を結び付けることは困難である。一方、教訓を特徴付ける属性の設定が困難であるため、分類問題に帰結することも困難であった。

そこで、問題解決手法として、具体的な事故から得られるキーワードを抽象化する連想ルールと、抽象化したキーワードと教訓を結び付ける関連ルールを記憶しておき、初心者がいくつかの具体的なキーワードを入力すると 関連するプログラミングノウハウを検索して表示するもの を採用した。

知識ベースの構築には 次の３段階を踏んだ。（ｉ）エキスパートから思い付くまま連想ルールと関連ルールを得る。（ii）初期化した知識ベースによる検索結果とエキスパートが判定した結果とを比較することにより ルールの洗練化を行う。（iii）ルールの保守ツールをエキスパートに提供する。

完全なルールは短期間では得られないが、エキスパートが気付いたときに新たなルールを入れれることが重要と考え、先に述べた知識ベースビューを（iii）の段階で設けた。採用した問題解決モデルはこのエキスパートシステムだけでなく他にも使えるので、設計した知識ベースビューは保守だけでなく別のエキスパートシステムの試行に利用できる。

（２）知識ベースを動的に保守する知識ベースビュー

ＭＥＡ（Means-Ends Analysis）に基づくプロブレムソルバを用いて、知識ベースを動的に保守するための知識ベースビューと先の４つの環境について論じる。ＭＥＡは初期状態と目標状態が与えられたとき、初期状態から出発し、現在状態を目標状態に近づくように状態を遷移するオペレータ（領域知識と呼ばれる）列を選ぶ手法であり、一般に、解の質と求解の効率を無視すれば、領域知識だけで問題を解くことができる [Newell, et al. 1972]。

質の高い解を効率良く得るためには、ある状態で複数のオペレータが候補となったときそのどちらを選ぶかの知識（戦略知識と呼ばれる）が必要となる。しかし、PRODIGY [Minton, et al. 1987] などで知られていた戦略知識はＭＥＡにおけるゴールスタックを追跡するものであり、複数のゴールが互いに干渉する場合には、それにとらわれない戦略知識が必要である。

ここでは、はじめに保守対象の知識となる新しい２種の戦略知識を示す。一つはゴールスタックに対し割込みをかけるもので、他はゴールスタックをクリアするものである。これらの戦略知識は机上で静的に獲得することは困難で、ソルバを動かし、問題を解いている途上、逐次動的に獲得されることを示す。

次に、これらの知識を獲得するには、開発環境でプロブレムソルバの機能を拡張する必要があることを示す。その後、新たに知識ベースビューを設けることによりエキスパートから戦略知識の獲得が可能となることを示す。さらに、戦略知識の獲得の効果をシミュレーション実験に基づいて述べる。

（３）表形式知識ベースとそれに対するデータベース（ＤＢ）ビュー

表操作を行うＯＡソフトウェアをベースとしたエキスパートシステムに関し、それを構築するための知識表現と知識ベースビューについて論じる。

本エキスパートシステムは、オフィスワークを状態遷移モデルとして捉え、予め登録しておいたオフィスワークの手続きをイベントドリブンに実行するものである。イベントとは、例えば、時間の到来、メイルの受信などである。自分の仕事の一部を代行させる秘書システムとみなすことができ、エキスパート＝エンドユーザである。

この種のＯＡシステムについては、従来、記述の簡易化と記述力の強化のバランスが課題であった。すなわち、ＯＢＥ(Office By Example)[Zloof. 1982] のように記述を簡易にすると登録できる手続きが単純となり、逆に SCOOP[Zisman. 1977] のように記述力を強化するとオフィスワーカが自らの知識を登録できないという問題があった。

ここでは、表操作を行うＯＡソフトウェアを利用できるオフィスワーカを想定し、処理対象物に状態という管理情報を持たせ、イベントと状態と処理の組をリレーション（関係）で表現することにより 記述力と記述の簡易性のバランスをとった。すなわち、表操作言語のカバーする範囲で記述力を確保し、ＤＢビューを知識ベースビューとして応用することにより 登録した知識へのアクセスを簡易にした。ここでの特徴は、表操作だけでオフィスワークの手続きを利用者主導で実行するか、システムにイベント起動で実行させるかを、適宜入れ替えることが可能なことである。

１．３　本論文の構成

本論文は、緒言と結言を含め ８章からなっている（図１）。

まず第２章は、本論文の準備であり、エキスパートシステムの基本概念・用語について説明し、エキスパートシステムの構築に関する従来の研究の成果について言及している。

次に第３章では、ビジネス分野における計算機応用システムの変遷について考察し、計算機部門による定形処理のプログラミング、エンドユーザ部門における非定形処理のプログラミングの両者を対比することにより、エキスパートシステムの関与者と構築手順を分析している。その結果、知識ベースの構築には、４種類の環境（試行環境、開発環境、利用環境、保守環境）が必要であることを導いている。それらの関係に関し、開発環境が基本となる機能を提供する環境と考え、知識ベースビューと呼ぶ一種のメタ知識を導入することにより、他の環境を統合してリサイクルできることを示している。

第４章から第７章は、具体的な知識ベースビューに関する研究を述べたものである。特に知識ベースの保守に関する知識ベースビューが重要と考え、（１）静的に（推論とは独立に）保守する場合（２）動的に保守する場合、さらに見方を変え（３）データベースの研究成果であるＤＢビューを用いた場合について、具体的な応用例に基づいて、その表現、機能、有効性、限界について述べている。

第４章は、静的に知識ベースを保守するための知識ベースビューに関するものである。知識ベースには、継続的に保守される領域知識と基本的に変更されない推論制御の知識があり、前者においても陽に定義できる構造があるとの仮定を設定し、知識ベースの構造と属性を表現するメタ知識を示し、さらにそのインタプリタの実現方法を述べている。このインタプリタは エキスパートに対する知識ベースの保守ビューとなり、新たな問題解決モデルが考案される都度生じていた 個別に知識獲得ツールを開発する労力を削減した。

第５章は、第４章で提案した保守ビューの 具体的な適用とその検証に関するものである。プログラミングノウハウを伝承するエキスパートシステムを例に、試行、開発、利用、保守環境が 静的な知識ベースの保守を実現する知識ベースビューにより いかに構築されるかを論じ、知識ベースビューが エキスパートシステムのアーキテクチュアを リサイクリックに利用可能とすることを示している。

第６章は、動的に戦略知識を獲得していくための知識ベースビューに関するものである。従来知られていた戦略知識では、質の良い解を効率良く求められないことを示し、新たな２種類の戦略知識を示した。続いて これらの戦略知識を静的に獲得することの難しいことを示し、第４章とは異なる 推論過程で戦略知識を保守するための知識ベースビューについて 述べ、エイトパズルの問題に適用したシミュレーションで その有効性を確認している。

第７章は、関係に基づく知識表現とそれに対する知識ベースビューについて述べている。オフィスワークを状態遷移モデルで表現したエキスパートシステムを例に、記述の簡易化と記述力の強化の均衡をとるために、知識をリレーショナルデータベースに蓄積することが有効であることを示している。

最後に、第８章では結言として、本研究で得られた成果を総括的に述べ、今後の課題を示す。

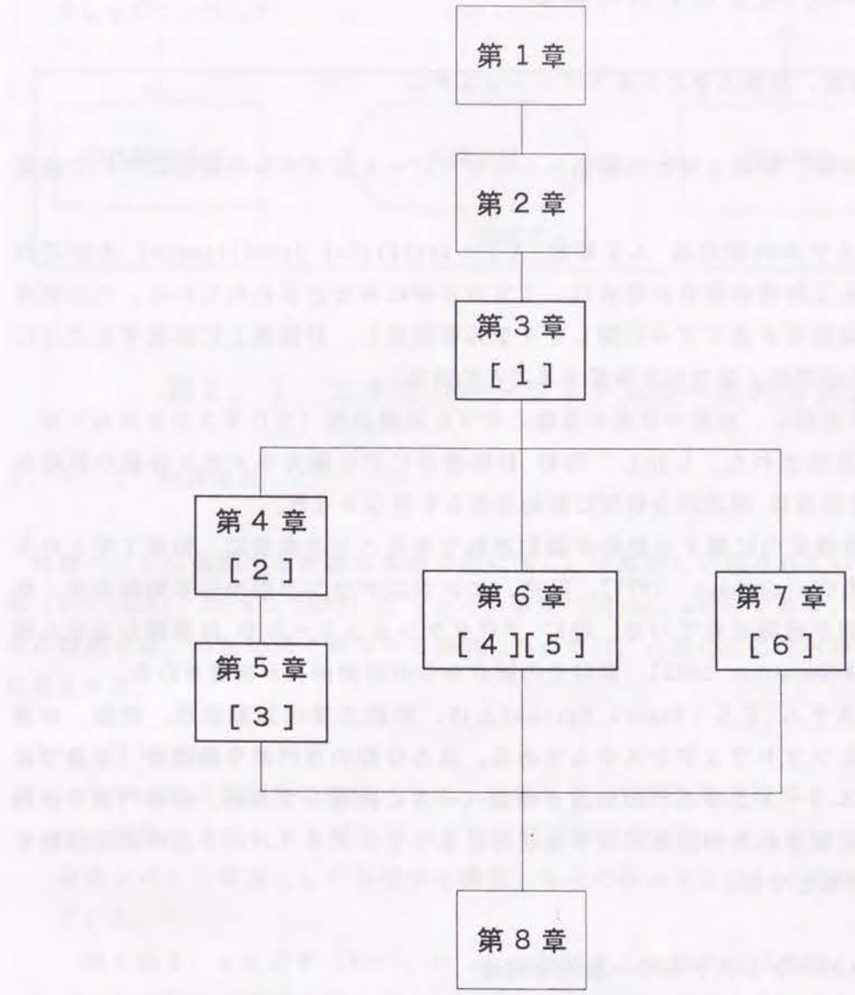


図１　本論文の構成

（注）［　］は、発表論文リスト（ｐ１４６）に掲載した論文の番号であり、
各章はその論文をもとに執筆したことを示す。

第２章　エキスパートシステム構築の課題と従来研究の流れ

２．１　エキスパートシステムの概要

２．１．１　人工知能、知識工学とエキスパートシステム

はじめに、人工知能、知識工学との関係からエキスパートシステムの研究について概観する。

エキスパートシステムの研究は 人工知能(ＡＩ：Artificial Intelligence) の研究の一分野を占める。人工知能の研究の原点は、１９５６年にあると言われている。この研究の目的は、人間の知能のメカニズムに関してモデルを設定し、計算機上に実現することにより、そのモデルの正当性・妥当性を検証することにある。

人工知能の研究の初期に、以後の研究の基盤となった知識表現（プロダクションルール、フレームなど）が提案された。しかし、当初 計算機のＣＰＵ能力やメモリ容量の制約から、これらの表現の応用は 原理的な研究に終始せざるを得なかった。

１９７７年、計算機能力に関する制約が緩和されてきたことを背景に、知識工学という考え方が提唱された[Feigenbaum. 1977]。以来、エンジニアリング面からも知識表現、推論、知識獲得の研究が展開されている。特に プロダクションルールで 計算機システム構成業務が記述され[McDermott. 1982]、実用化の観点からの研究開発が加速された。

エキスパートシステム(ＥＳ：Expert System)とは、知識工学の知識表現，推論、知識獲得技術を応用したソフトウェアシステムである。ある分野の専門家や熟練者（本論では以下総称してエキスパートと呼ぶ）の知識を知識ベースに記憶しておき、非専門家や未熟練者が、計算機に記憶された知識を利用することによって、エキスパート並の知的活動を可能にすることを目的とする。

２．１．２　エキスパートシステムの一般的な構造

エキスパートシステムは 専門知識を取り扱うシステムである。専門知識とは 例えば、

（１）仮説を設け 代替案を作成しながら 評価する、
（２）許容時間内に求解するために 戦略を使用する、
（３）過去の経験を利用して 問題解決の効率化を図る、
（４）不確実なデータ、知識を扱う、

などの知識を含む。

これらの知識を計算機で処理するには、それらの表現手段、記憶手段、利用（推論）手段、獲得手段が必要である。エキスパートシステムの一般的な構造を図２．１に示す。図のナレッジエンジニアについては ２．２．１で述べる。

エキスパートシステムの構造は、知識を取り扱う新しい価値以外にもソフトウェア開発

の新しい手法としての価値が指摘されている。これについては、２．１．５で述べる。

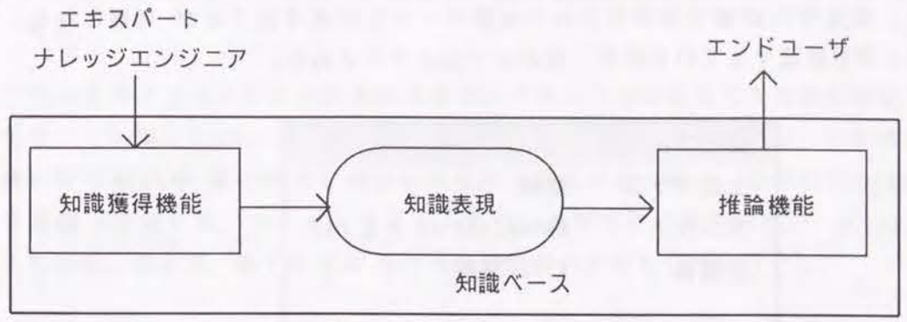


図２．１　エキスパートシステムの一般的な構造

２．１．３　知識表現

知識ベースに格納する知識の表現方法に関し、一般的に利用されているのは、ルール表現 [Kobayashi, et al. 1985] とフレーム表現 [Ogawa. 1985]であり、これらについて簡単に説明する。以後の章で利用する表現については、必要に応じて付録などを用いて詳細に言及する。

（１）　フレーム

フレームは事実型の知識（もの・対象とする世界）を表現する。一つのフレームには、複数のスロットと呼ぶデータ格納欄が設けられる。さらに、フレームは ものの抽象レベルを階層化して表現する機能、ものの構成を分割して表現する機能を具備している。

例を図２．２に示す [Mori, et al. 1988]。この図では「会社」フレームの下位に「株式会社」「有限会社」フレームがあること、「株式会社」フレームの下位に「Ａ社」「Ｂ社」フレームがあることを示している。また、「会社」フレームには「ランク」「従業員」スロットがあり、それらのとりうる値や属性が定義されている。

フレームの中には、「デモン（付加手続き）」と「メソッド」と呼ばれるプログラムを記述することが可能である。

デモンは、フレーム中のあるスロット値が更新あるいは参照された時に起動されるプログラムであり、データ駆動型の推論に使用される。例えば、温度のスロットに対し、温度計メモリ表示プログラムを付加することにより、温度スロット値が更新されたときにメモリ表示を連動して変化させることが可能である。

メソッドとは、オブジェクト指向言語 [Goldberg, et al. 1983] における用語である。オブジェクト指向とは、データと それに対する処理手続き（メソッド）の組を単位としてプログラムを定義し、それらの間のメッセージ（送り先のオブジェクト

に対する処理依頼）交換によって、所定の処理を行わせるものである。フレームのデータ値を参照あるいは更新する場合、必ずそのフレームに付随しているメソッドを用いる必要があるため、フレームが 内部情報に関して相互に独立した知識モジュールとなり、変更時の影響が局所化された知識ベースを作成することが 可能となる。本論の第４章で提案するＦＯＲＭも一種のオブジェクトである。

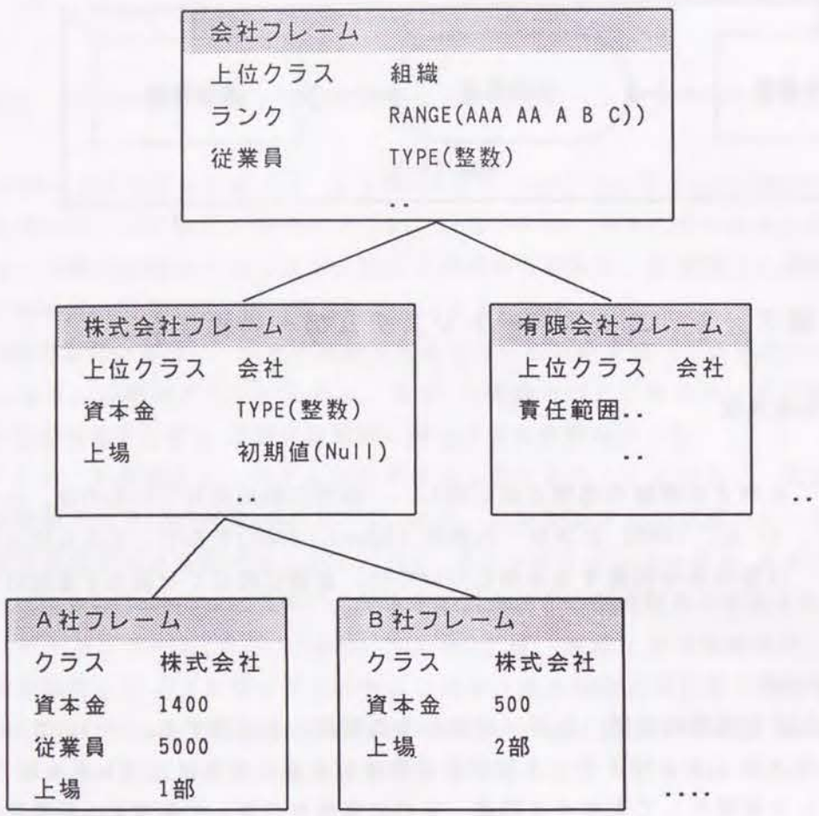


図２．２　フレーム表現の例

フレームには階層関係があり、各フレームは複数のスロットからなり、各スロットはスロット名とスロット値からなる

（２）　ルール

ルールは、エキスパートの経験や手順（オペレーション）に関する知識を表現するために使用され、「ＩＦ（条件）ＴＨＥＮ（結論）」の形で記述する。条件から結論，条件から行動，原因から結果などの関係を表現する。この種のルールを利用して推論するシステムを「プロダクションシステム」という。ルールの例を図２．３に示す[Mori, et al. 1988]。この例では、条件・結論が（フレーム名）、（スロット）、（スロット値）の３つ組で表現されており、？記号はそれが変数であること、＠記号はそれがスロットであることを示している。（これらの形式的定義については各章にて必要に応じて説明する）

ルールは 大きく２種類に分類される。一つは 領域知識あるいはオペレータと呼ばれるもので 対象としている世界（あるいは状態空間）を更新する。もう一つは 複数のルールの条件部が満たされるとき どのルールを選択するかに関するメタな知識で、これは 戦略知識と呼ばれる。後者は、競合解消に関するもので ２．１．４でも言及する。

フレームのメソッドにルールも記述させ、フレーム中心に全ての知識を記述する試みも報告されている[Chusho, et al. 1989]。ルールについては、２．１．４で示すように、本論の第４章以降 第７章までのあらゆる章で言及する。また、ＰＲＯＬＯＧなどで使われる述語（関係）は、ルール、フレームを共に表現できる体系[Tsuji, et al. 1987d]を与えており、第６章、第７章では 述語を知識表現の手段として利用する。

```
信用度チェックルール
    IF      ?会社 の @資本金 が 1000 以上で
                     @上場   が 1部 または 2部 である
    THEM    ?会社 の @ランク を AAA とする

融資額設定ルール
    IF      ?会社 の @ランク を AAA で
                     @従業員 が 1000 以上である
    THEM    ?会社 の @限度額 を 10億円 とする
```

図２．３　ルール表現の例

この例では、条件部・実行部とも対象世界の（フレーム名、スロット名、スロット値）の組で表現されている

２．１．４　推論

推論にも 多様な形式のものが提案されている。ここでは ルールに基づく推論機構の基本的な動作を説明する。詳細な動作定義については、各章の必要に応じて言及する。

（１）前向き推論

条件部が成立するルールを逐次実行し、結論を得るものである。条件部の成立するルールが皆無になった時点で動作を終了する。前向き推論の応用については、本論の第５章、第７章で述べる。

（２）後向き推論

目標（ゴール）を与え、その目標を導く結論部をもつルールの条件が成立するか否かを検証する。ある目標を導くルールの条件が成立するか否か不明の場合、その条件をサブゴールとして、同じ動作を繰り返す。後向き推論の応用については、本論の第６章で述べる。

前向き推論は「状況対応型」、後ろ向き推論は「目的指向型」とみなすことができる。一つのエキスパートシステムを実現するのに 前向き推論と後ろ向き推論の両者を組合せることも可能である。第６章ではこの組み合わせについても言及する。

（３）競合解消

推論を行なう上で複数のルールが実行可能となる場合がある。このとき、どのルールを最初に実行するかを決定する必要がある。この動作を「競合解消」という。問題を適切に解くか否かはこの競合解消戦略に依存する。

競合解消の方針(戦略)は、エキスパートシステム構築時に決定され、基本的に変更されることがない場合と、構築時には決定することができず、問題解決のトレースをみて段階的に決定される場合がある。本論文の第４章の方式は 前者を前提としており、第６章の方式は 後者を前提としている。

２．１．５　知識獲得

知識獲得には、推論と独立に静的に獲得する場合と、推論に依存して動的に獲得する場合とがある。さらに、推論のための知識をそのまま獲得する場合と、例を獲得し推論のために帰納あるいは演繹し学習する場合とがある。このマトリックスを表２．１に示す。

表２．１　知識獲得の分類

獲得の	知識の与え方	
タイミング	直接編集	例からの学習
推論と独立	４，５，７章	本論の
推論に従属	６章	範囲外

本研究は この知識獲得の一分野に関するものである。本論は、学習ではなく知識を直接獲得する方式に関するものであり、第４章、第５章、第７章は 静的に獲得する方式、第６章は 動的に獲得する場合を論じる。

知識獲得は広義には、専門知識のモデル化の研究を含み、エキスパートシステム構築の研究＝知識獲得の研究と言っても過言ではない。

２．１．６　構造から見たエキスパートシステムの特徴

エキスパートシステムの構造が 図２．１を呈していることから、従来のソフトウェア技術では取り扱いが困難であった 次の課題を解決すると期待されている。逆に エキスパ

ートシステムは これらの期待を満足するように開発されるべきである。

（１）ラピッド・プロトタイピング

従来、ソフトウェア開発のためのライフ・サイクルの考え方が提唱され、要求分析を正確にすることが重要視されてきた。要求分析は、銀行のオンライン・システムなどの大量の定型的なデータ処理システムに対しては 可能であるし、必要とされてきた。しかし、最近 要求分析が正確に実施できるという前提を放棄しなければならない分野が数多く存在することが 判明してきた。この分野の応用システムの開発にはプロトタイプ・システムを短期間で作成し、そのシステムを評価しながら実用化を指向していかなければならない。

図２．１に示すように、エキスパートシステムには 推論機構が予め備え付けられている。そのため、一部の知識を記述するだけで局所的な動きが実現され、ラピッド・プロトタイピングに適している。

（２）大規模ソフトウェア開発

我々の開発する応用システムは 日々大規模化の傾向にある。これまで、機械語 → アセンブラ語 → 高級言語という言語記述力の発展により、一人当たりのプログラム開発量が 増加してきた。図２．１に示す知識ベースに蓄積される知識を表現する手段は、これをさらに一歩推進したものと考えることができる。すなわち、知識表現と呼ぶ新しいプログラミング言語は、より人間の思考に近い表現法を与えるため、より大規模、複雑なソフトウェアを実現できる可能性が存在する。

（３）計算機非専門家によるプログラミング

多くの組織において、システム開発の要求量は システム開発能力を上回っている。そのため、バックログ（ソフトウェア開発の積み残し）が増大している。さらにシステム開発部門が保守対象とするソフトウェア量も その保守能力を超えつつある。

これらの問題に対する解決策は、計算機部門以外のユーザによるプログラミングを可能にすることである。エキスパートシステムの知識を エキスパートから直接 計算機に入力することが可能となれば、プログラマなしのソフトウェア開発が実現される。図２．１の知識獲得機能は、この要求に応えるものとして期待されている。

（４）情報資源管理

データを組織の財産としてデータベースに収集し、それを分析することにより種々の管理を行なっている事例が増加している。図２．１に示す知識ベースはこの枠組みを拡げ、組織の中に散在する知識を収集し、それを財産として活用することを可能とすると期待されている。

これらの期待の実現には、エキスパートシステムにまつわる知識工学の技術分野を越えて幅広い議論を必要とするが、以上の意味を考慮したエキスパートシステムの構築に関する研究は重要なものであろう。

２．２　エキスパートシステム構築のモデルと手順

２．２．１　エキスパートシステム構築の関与者

エキスパートシステムの構築には 次の人々が関与する。

（１）エキスパート（知識の提供者）

（２）エンドユーザ（システムの利用者）

（３）ナレッジエンジニア

ところが、従来のエキスパートシステム構築に関する研究は 知識表現、あるいは知識獲得など 知識を計算機に移植するための技術課題に重きをおくあまり、誰がどうエキスパートシステムの構築に参加し、同一の知識ベースに対して、誰がどう見るかなどの運用面の課題について 未解決であった。これらの関与者のエキスパートシステムの構築における役割分担については、第３章で分析する。

ここでは、ナレッジエンジニア(ＫＥ：Knowledge Engineer) について述べる。

ナレッジエンジニアは、専門家にインタビューして その知識を抽出し 分析して，知識ベースを構築する役割をもつ。設計者としてエキスパートシステムの目的，システム化の範囲，機能を設定し，実際の開発プロジェクトの推進に関する管理，統制を行う。

ナレッジエンジニアの存在価値は、次に述べる専門家モデル・知識モデルの分析能力にある。モデルを決定して、ある領域のエキスパートシステムとして必要な知識ベースを設計する。そのため、ナレッジエンジニアは、システムエンジニアの技術に加え、次の４つの知識、技術を必要とする。

（１）人工知能，知識工学の知識

（２）エキスパートシステムを構築するためのツール及び知識表現言語の知識

（３）業務知識（専門家と議論でき、その分野での知識の構造を決定する能力）

（４）インタビュー技術

２．２．２　専門家モデルと知識モデル

情報処理システムの構築は、ある業務（what）をプログラム（how）に変換することであると考えられる。しかし、この変換を直接行なうことは 困難である。そのため、エキスパートシステムに係らず、従来の情報処理システムの設計においても、現実の業務に対し コンピュータ化する範囲をモデル化する段階をおく。従来の設計におけるモデル化では、帳表、台帳、伝票などデータを中心に それらの関連を分析し、どの範囲の業務を計算機化するかを決定するのが 課題であった。

それに対し、エキスパートシステムの設計では、エキスパートの行っている問題解決法、すなわち、知識中心にモデルを作成する必要がある。データと知識の区別は 明確ではないが、一般には、データは「それがもつ値そのもの」に価値があると考え、知識は「推論により使われて初めて価値が出るもの」と考えられる。モデル作成とは次のことを言う。

（１）現実のエキスパートの行動を近似する。

（２）扱う問題の範囲を限定する。

（３）計算機化する部分を明確化する。

コンピュータ化する範囲のことを「概念世界」と呼ぶと、図２．４のようなダイアグラムができる［Kataoka．1988］。すなわち、エキスパートシステムの構築作業は、

①実世界のｗｈａｔを概念世界のｗｈａｔに変換し、

②概念世界の中で、ｗｈａｔからｈｏｗを導き、

③概念世界のｈｏｗを実世界のｈｏｗ（すなわち知識ベース）に具体化する、

からなると考えることができる。

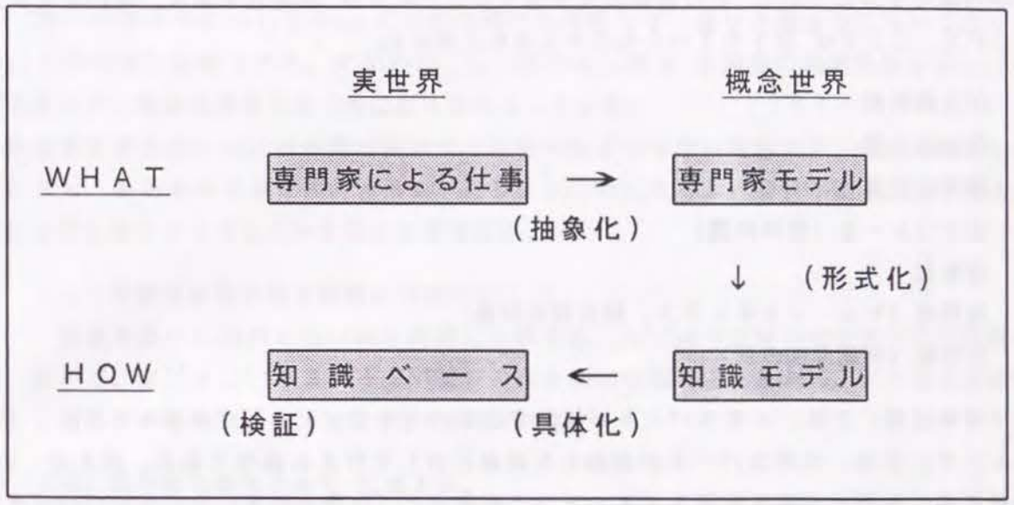


図２．４　エキスパートシステムの構築作業

実世界で、直接知識ベースを構築することが困難であり、モデル化の手順を踏む

本論では、概念世界におけるｗｈａｔを「専門家モデル」と呼び、ｈｏｗを「知識モデル」と呼ぶ。前節で述べたルールやフレームは 知識モデルである。このダイアグラムにおいて、

・実世界のｗｈａｔ（すなわち専門知識）から概念世界の専門家モデルを作成することを「抽象化」、

・専門家モデルから知識モデルを導出することを「形式化」、

・知識モデルをもとに概念世界のｈｏｗ（すなわち知識の記述）に写像することを「具体化」

・知識ベースの正しさを確認することを「検証」

と呼ぶ。ただし、抽象化と形式化の境界、形式化と具体化の境界は必ずしも明確ではない。また、抽象化、形式化、具体化、検証は、システム開発の順序ではなく、作業内容を表すものである。

専門家モデルの考察により、類似問題を全く別の手法で解いていたり、全く異なると思える問題が同じ解法のバリエーションに他ならないことがわかる。相続相談、年金相談といった「相談」というキーワードから表層的に類似していると思われるエキスパートシステムの専門家モデルが全く異なっていたり、年金相談、故障診断といった表面からは似ていると思われないエキスパートシステムの専門家モデルが 同一であることもある。[Chandrasekaran. 1985]。

2．2．3　専門家モデルの枠組み

専門家モデルについて その枠組みを提示することにより 必要性を示す。専門家モデルについて、ここでは 以下の７つのものからなると考える。

①目標状態
②初期状態
③対象世界
④オペレータ（領域知識）
⑤仮説
⑥戦略（ヒューリスティクス、競合解消知識）
⑦経験（問題解決の例）

「対象世界」とは、エキスパートが認識するもの（オブジェクト）の集合である。「オペレータ」とは、エキスパートが認識した対象に対して行える操作である。例えば、「旅費精算書に旅費と出張日を記入する」という事象では「旅費出張書」というオブジェクトと「記入する」というオペレータがある。旅費と出張日の記入欄は、旅費精算書の中にあると考える。オペレータのことを 領域知識ということもある。

「状態」は、オブジェクト（あるいはその集合）に対して付与される。例えば「旅費精算書」に対して「白紙」という状態と「記入済み」という状態があるとすると、先のオペレータにより状態が更新される。「問題を解く」ということは、対象世界の状態を「初期状態」から「目標状態」へ「オペレータ」を用いて遷移させることである。

エキスパートシステムの成否・良否は その専門家モデルの仮説，戦略，経験，不確実性の扱い方に依存する。エキスパートによる問題解決の特徴については、以下のように考えられる。[Hayes-Roth, et al. 1983]

（１）仮説を設け 代替案を作成しながら 評価する。

この問題解決法は、生成検査法(generate and test)と呼ばれる。仮説の候補がすべて生成できる仕組み，同一仮説を２回以上生成しない仕組みが必要である。一般に仮説候補の生成法に知識を利用することにより、不要な推論が削減され、求解効率がよいことが知られている。

（２）許容時間内に求解するために 戦略を使用する。

エキスパートシステムは、問題を解くために多数の組合せを評価しなければならないことが多い。組合せが多数の場合、最適解を求めることは時間的に不適切である。例えば2^{10}通りの組合せについて０．００１秒で解ける問題も2^{40}通りの組合せについては１０日以上の時間を必要とする。すなわち、しらみつぶし法は 小規模の問題を解くことに有効であるが、複雑な問題を解く時には不適切なことが多い。

最適解を求めるためには多数の組合せを評価せねばならない問題でも、観点を変えることにより、ある条件をみたす許容解を求めるように考えるとよい場合がある。以下では、その見方に関するエキスパートの技法例を示す。

（ａ）問題を複数の部分問題に分割する。

問題を互いに関係しない部分問題に分割する。2^{40}通りの組合せが発生する問題において、$2^{10}+2^{10}+2^{10}+2^{10}$通りの組合せに分割可能であれば、１０日以上必要とされた求解時間が０．００４秒に削減される。

（ｂ）部分解を順次求めて 拡張する。

部分問題の解が 全体の解の一部になるかどうか 判定できない場合、（ａ）のような単純分割はできない。設計／計画の問題の多くは 独立な部分問題に分割出来ないため、順序付けた部分作業に分割し、部分構成を作り出しては それを拡張することにより組合せの発生を押える。例えば、レイアウト問題では、あらゆる組合せをしらみつぶし的に評価するのではなく、基準となるものから順次配置する（部分構成を作り出しては拡張する）ことにより、効率をあげることが可能である[Watanabe,et al. 1985]。

（ｃ）トップダウンで問題を解く。

問題解決が 大局的な部分作業から局所的な部分作業に順次分割される場合、より大局的な部分作業を先に解いた方がよい場合がある。１つの大局的な部分作業が解けた結果として、複数の新しい局所的な部分作業が生じる。この種の問題については、各部分作業に対して 解く順番を与えることにより、組み合わせの発生を押さえることができる。例えば より優先度の高い制約だけを考慮して問題を解き、順次制約を追加していく方法が考えられる。これらの手法は、トップダウン改善法と呼ばれ、従来行われてきたウォータフォール型のソフトウェア設計には これが使用されている。

（d）解いている途中に 優先順位を変更する。

（c）で述べた順位を 事前に決定できない場合がある。問題解決の途中に得られる情報で優先順位を変更しなければならない場合、次の知識と推論メカニズムを用いて組合せの発生を抑制する。

・優先順位を変更する知識

・優先順位が変化した時、一時的に推論を中断するメカニズム

・中断していた推論を再開するメカニズム

この手法は 最小拘束原理と呼ばれ、リアルタイムの監視，制御のエキスパートシステムなどで使用される。

（3）過去の経験を利用して問題解決の効率化を図る

エキスパートが問題を解く時、1から問題を解くことは少い。使用できるものがあれば、過去の類似の事例を検索してそれを流用することにより効率よく問題を解いている[Shank. 1982]。これについては、第6章で触れる。

（4）不確実なデータ，知識を扱う

エキスパートは、問題解決に必要十分な情報を利用できないことがある。例えば、

短時間に然るべき判断を下さなければならない場合、

関連するデータが部分的に得られない場合、

データ値を信用できない場合、

データを解釈するための知識が信頼できない場合、

などである。不確実なデータ，知識を扱うモデルの例として 次をあげることができる。

（a）連想

ある仮説が成立していることから、別の仮説も成立するであろうという知識を使用して、データの不完全性を補う。例えば、「消防車」と「サイレン」から「火事」を連想できる。「消防車」あるいは「サイレン」だけから「火事」は直ちには関連付かないが、それらが組み合わさると関連付く。検索を行う場合など、あるキーワードが指定された時、別のキーワードも含めた方がよい場合に利用する。これについては第5章で論じる。

（b）得点（スコアリング）

複数の代替案（例えば、意思決定の相談や故障の原因究明など）の順序付けを行うものである。個々の現象から原因としての可能性の得点を与える知識や、相談者のもつ属性から行うべき行動の優先順位を与える知識に用いる。例えば、顧客の要求に答える自動車を推薦する問題において、価格、定員、用途、維持費、などそれぞれの項目について要求の適合度を得点として付与し、最終的に総合評価を下すなどである。

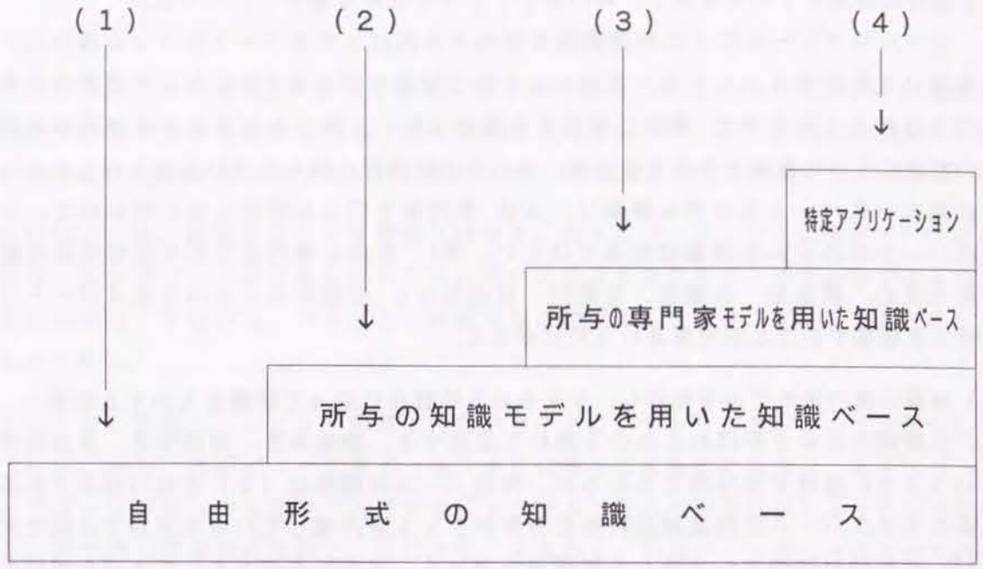


図2．5　エキスパートシステムの構築方法からみた
専門家モデルと知識モデル

（c）確信度

「AはBを暗示する」「CとDなら Eではなさそうだ」などの漠然とした判断を表現するための手段であり、便宜的な数値により意味表現を近似的に表す手法である。例えば、1を真、－1を偽とし、「AならばBは0．7位確かである。」「CとDではEは、－0．6位 確かである」というように扱い、必要に応じて経験的に与えられる閾値と比較して推論を行う。

2．2．4　エキスパートシステムの構築方法からみた専門家モデルと知識モデル

先に述べた専門家モデル、知識モデルの関係を基に分類すると、エキスパートシステムの作成方法には 次の4つのレイヤがある。これらの関係を 図2．5に示す。

（1）知識モデルから作成する場合

知識モデルとして、既知のルールやフレームでは不適当な場合、新たな知識モデルと それに対応する推論メカニズムを ＰＲＯＬＯＧやＬＩＳＰなどの知識処理言語を用いて開発する。高度な推論方法を実現あるいはより高速性を指向した推論メカニズムを実現する必要がある場合、このレイヤから構築を行う。

（2）既存の知識モデルを使用し、専門家モデルから定める場合

ルールやフレームなどの知識表現を提供する汎用エキスパートシステム構築ツールを用いて記述する。ルール、フレームという知識モデルとそれに対応する推論メカニズムは与えられるので 個別に作成する必要はない。ルールとフレームが大半の知識の記述に十分な表現力を示すことは、これまで経験的に知られている。しかしながら、汎用エキスパートシステム構築ツールは 専門家モデルを明示していないので、エキスパートシステムの構築は容易ではない。逆に 自由に専門家モデルを定式化可能であるため、相談型、分類型、診断型、計画型など 任意のタイプのエキスパートシステムを構築することが可能という利点がある。

（3）既存の専門家モデルを使用し、与えられた枠組みに沿って知識を入力する場合

分野別シェルと呼ばれるものを用いて記述する。検索向き、相談向き、設計向きというように適用対象は限定されるが、知識ベースの構築は（2）と比べ容易である。あるエキスパートの問題解決行動を分野別シェルが内蔵しているモデルで近似できれば、比較的短期間でシステムを開発可能である。専門家モデルとして十分な種類のものを用意できれば、エキスパートシステムの構築は飛躍的に容易になると予想されている[Chandrasekaran. 1985]。

（4）既存の知識ベースを修正して使う場合

完成した知識ベースの一部の知識を修正したり、追加するだけで新たなエキスパートシステムを構築する場合がある。短時間でエキスパートシステムとは何であり、何ができるのかを認知するときにも有効に利用できる。

2．2．5　知識ベースビュー

知識ベースビューとは 本研究で新たに提案する概念であり、第3章でその必要性を論じ、以後の章でその具体化を図る。この知識ベースビューは、データベース（ＤＢ）技術のビューという概念 [Date. 1981][Ullman. 1980] の影響を受けている。

ＤＢビューは、仮想的なデータ構造（利用者に対するデータベースの見え方）を定義する手段、エンドユーザが意識しなくてよい部分にマスクを与えデータを保護する手段、などを提供する。すなわち、ＤＢビューは利用者に対するヒューマンインタフェースを与えるものである。

それに対し知識ベースビューの基本的なアイデアは、ナレッジエンジニアが開発した知識ベースに対し、知識ビューと呼ぶ一種のメタ知識を与えることにより、エキスパートやエンドユーザが彼らの抽象レベルで知識にアクセスできる手段を提供しようというものである。概念的な位置付けを図２．６に示す。すなわち、知識ベースビューはエキスパートシステムのエキスパートやエンドユーザに専門家モデルを提示するものである。

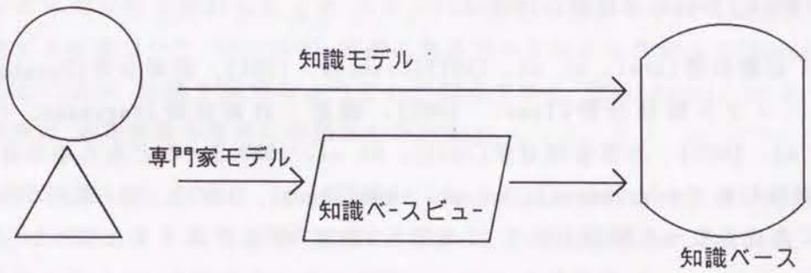


図２．６　知識ベースビューの位置付け

知識ベースビューを介すと、記憶されている形式より抽象度の高い記述形式で知識ベースにアクセスできる

２．３　エキスパートシステムの構築に関する従来の研究

本節では、これまで行われてきたエキスパートシステムの構築に関する研究の概観を述べる。

２．１節でも述べたが、エキスパートシステム研究の原点は、１９５６年に行われたＡＩの会議にあると言われている。当初、代表的な知識表現であるプロダクションルール[Simon, et al. 1972] [Kobayashi, et al. 1986]、フレーム[Ogawa. 1985]などの知識表現や推論機構の研究が活発に行われ、米国内ではＡＡＡＩ主催の会議、国際的にはＩＪＣＡＩという会議が定期的に開催されるとともに Artificial Intelligence, AI Magazine という刊行物が発行されるようになった。

１９７７年、[Feigenbaum. 1977]により、「知識表現や推論機構の優劣より、対象分野の知識の優劣が、システムの能力を決定する」と指摘され、「Knowledge is Power」という考え方が 注目されるようになった。以来、知識工学という名のもとで、エンジニアリング面からも研究がなされるようになり、わが国でも、情報処理学会、１９８６年に設立された人工知能学会などを中心に幅広く研究がなされている。１９８９年には、ＩＥＥＥから Knowledge and Data Engineering という論文誌が刊行され、知識工学はそのエンジニアリングとしての地位を確立したように思う。

エキスパートシステムは、ある特定分野の専門知識を用いて、ハイ・パーフォーマンスな問題解決を行うシステムである。特に、[McDermott. 1982]によりプロダクションルールで計算機システム構成業務が記述され、同様のシステム開発が相次いで行われた[Wu, et al. 1986] [Tsuji, et al. 1987d]。

以来、診断・分類分野[Arai, et al. 1987][Clancey. 1984]、制御分野[Funahashi, et al. 1987]、ソフト開発分野[Tamai. 1987]、設計・計画分野[Nagasawa. 1987] [Watanabe, et al. 1985]、品質管理分野[Tsuji, et al. 1988a]、などあらゆる分野でシステム開発が行われている[Sasaki, et al. 1986][Hirai. 1987]。さらに エキスパートシステムに焦点を絞った雑誌として ＩＥＥＥ ＥＸＰＥＲＴが１９８６年から刊行されている。

当初のエキスパートシステムは、ＬＩＳＰやＰＲＯＬＯＧなど記号処理言語により開発されることが多かった。その後、 KEE[Fikes, et al. 1985] や ES/KERNEL[Kanamori, et al. 1987][Yoshimura. 1988] のように、ルール、フレームなどの汎用的な知識モデルを具備するエキスパートシステム構築ツールが普及し、これらを使用して開発されることが多くなっている[Barstow, et al. 1983]。最近、これらの構築ツールは 知識エディタ[Kumagai. 1988]、デバッガ、ブラウザ[Conklin. 1987]、ファイルシステムなどが統合されたプログラミング環境[Barstow, et al. 1984] [Tsuji, et al. 1989b]として充実しつつあり、エキスパートシステムの構築は 一段と加速している。ツールに

関してもＩＥＥＥ で Tools for Artificial Intelligence という国際会議が定期的に開催されるようになっている。

構築ツールと組として使われるエキスパート構築技法についても、いくつかの発表[Hayes-Roth, et al 1983][Schoen, et al. 1987][Hanaoka, et al. 1990] がなされている。それらはいずれも、①プロトタイピングの重要性（従って初期知識の作成とそれらの洗練化）、②初期知識の作成は問題解決のモデリングと知識表現への定式化からなること、③知識の洗練化には静的なデバッグと動的なデバッグがあること、を言っている。平成３年の人工知能学会全国大会では、いくつかの手法が特集として発表されたが、今のところ提案されている手法に決定的な差を見出すことはできない。

エキスパートシステムの構築では、「知識獲得のボトルネック」という言葉で知られるように、いかなる知識をいかにして収集するかが大きな課題である。ところが、先に述べたような汎用的なエキスパートシステム構築ツールは、どのような問題領域に適用できるのかが明確でないという指摘がなされている[Chandrasekaran. 1985][McDermott. 1986][Clancey. 1984]。

この問題に対して、[Chandrasekaran. 1985] により generic task と呼ぶ 抽象度がエキスパートに近い問題解決手法を見出すことの重要性が指摘された。そして、ある程度 問題解決のタスクを絞った知識獲得ツールとして診断型エキスパートシステム向けのＭＯＬＥ [Eshelman, et al. 1986] 、分類型エキスパートシステムＣＳＲＬ [Bylander, et al. 1983] 、計算機システム構成エキスパートシステム向けのＳＥＡＲ [van de Brug, et al. 1986] などが 開発されてきた。これらは、診断や分類など応用分野を制約した上で エキスパートの問題解決手法を個別にモデル化し、そのモデルに基づいて ツールが 直接エキスパートにインタビュー[Kawaguchi. 1989] することにより 知識を獲得しようという試みである [Mizoguchi, et a. 1988]。これらの研究は 初期知識の作成に有効なものである。

知識獲得については、Knowledge Acquisition Workshop が 定期的に開かれ、１９９０年には日本で開催された [Boose. 1990][Gaines, et al. 1990]。また、１９８９年より、Knowledge Acquisition という雑誌が Academic Press から定期刊行されている。

知識ベースの構築に関しては、初期知識の獲得だけでなく、陳腐化した知識の除去や新たに得られた知識の追加という保守の問題がある [Tsuji, et al. 1987b, 1988b]。この知識ベースの保守の重要性は「エキスパートは 自分が思っている以上に知っている」という名言で知られている。

知識ベースの保守を強調したツールとして、TEIRESIAS[Davis, et al. 1982] がある。TEIRESIAS は、知識の構造に関する知識をメタ知識と呼び、これを用いて知識ベースの保守を実現している。本システム は診断型のエキスパートシステムであるＭＹＣＩＮ

のフロント・エンドとして インプリメントされている。保守を重視した他のシステムには、正例と負例を与えて、それを性能良く分類するように ルールを逐次洗練していく SEEK2[Ginsberg, et al. 1985] などが知られている。

知識獲得の別の側面として、機械学習の話題がある。機械学習とは、専門家から知識を獲得するのではなく、システムが自動的に知識を獲得することである。先の SEEK2も例からルールを確定するので一種の学習システムである。機械学習には 大きく帰納学習と演繹学習がある。

帰納学習については、 ID3[Quinlan. 1983]、バージョン・スペース法 [Mitchell, et al. 1983] などが代表的なこれまでのアルゴリズムである。前者は 問題解決例からエントロピーを用いて初期知識（分類木）を学習するものである。後者は あるオペレータの適用に関し正例を損じない境界と負例を混入させない境界を押さえ、正例と負例を与えていき、それらの境界を順次収束させていくものである。帰納学習では、基本的に多数の事例から 知識を獲得する。

演繹学習は、領域の知識を用いて 問題解決のトレースを説明することにより 知識を獲得するので、特に説明に基づく学習 [Mitchell, et al. 1986][Dejong, et al. 1986] とも呼ばれる。演繹学習では、一つの事例からも知識を獲得することができる。

機械学習については Morgan Kaufmann より３回にわたり、単行本 Machine Learning [Michalski, et al. 1983] が出版されているほか、Machine Learning という雑誌も Kluwer Academic Publisers から定期刊行され、同名のワークショップが米国で毎年開催されている。

知識獲得として 頻繁に研究の対象となるのは、探索のためのヒューリステックに関するものである [Nilsson. 1980]。Means-Ends Analysis (MEA) [Fikes, et al. 1971] [Newell, et al. 1972] は、基本的な探索手法である。

ＭＥＡの探索効率を改善するために、これまでに いくつかの手法が 提案されている。例えば、現在の状態と目標状態の距離（解の長さ）を短くするための マクロオペレータ[Fikes, et al. 1971]、組み合わせ爆発の抑制に有効な制御知識を用いる方法 [Minton. 1988][Mitchell, et al. 1983] などがある。

近年、過去の問題解決の事例を再利用すること [Shank. 1982] が、組み合わせ爆発を抑制しつつ、解の長さの短縮に貢献すると期待されている。これは事例ベース推論 [Bradtke, et al. 1988] [Hammond. 1986] [Kolodner. 1985] [Ruby, et al. 1989] として知られている。著者等も 最近この分野の研究に着手しており、スーパコンピュータのチューニング問題への応用 [Akifuji, et al. 1990a 1990b 1991]、レイアウト問題への応用 [Yoshiura, et al. 1990]、プロジェクト管理への応用 [Taniguchi, et al. 1991] などで その有効性を示している。

事例は 一般的に利用できるルールあるいは規則に対して 浅い知識と呼ばれることがある。一般には、二つの知識の記述レベルを比較して 一方が他方の正当性を説明するとき、前者の方が後者より「深い」ということが多い。

そのため、ルールや規則だけでなく、深い知識（原理原則に近い知識）を組み合わせて知識獲得を行いながら問題解決をはかろう という研究もなされている [Ueno, et al. 1988]。特に、算術式をもとに推論を行うことを定性推論[de Kleer, et al. 1985] と呼び、ビジネスシミュレーション[Kosy et al. 1984] [Apte, et al. 1986] などへの応用が検討されている。

エキスパートシステムの知識ベース構築を エンドユーザプログラミングの延長で捉えることもある。これは、ＡＣＭの Transaction On Office Information Sytems や International Journal of Man-machine Studies で論じられており [Hewitt. 1986] [Winograd. 1988]、オフィスワークの記述に関するものが 活発に発表されている [Suchman. 1983]。

例えば、データベース、ワード・プロセッシング、電子メイル、トリッガ・プログラムを統合したＯＢＥ [Zloof. 1982]、フォーム（定型用紙）に基づく半構造的な仕事を自動化するシステムＦＯＲＭＡＬ[Shu, et al. 1982][Shu. 1985]、ＯＦＳ [Tsichritzis. 1982]、ペトリネットとプロダクションルールでオフィスの手続きを記述するＳＣＯＯＰ[Zisman. 1977] などが知られている

この分野は、オフィスワークが 元来単独の個人ではなく 複数人のグループにより行われること[Greif. 1988] [Suchman. 1989] [Ishii. 1989]、そのため知識が分散すること[Smith, et al. 1981]、などにより 分散した知識ベースの研究が必要となりつつある。

第３章　知識ベースビューの導入によるエキスパートシステムの構築モデル

３．１　まえがき

エキスパートシステムは、当初 知識モデルから 個別に設計し、ＰＲＯＬＯＧやＬＩＳＰなど 記号処理言語により 開発されることが多かった。その後、ルール、フレームなどの汎用的な知識モデルを具備するエキスパートシステム構築ツールが普及し、これを用いて開発されることが 多くなっている。このエキスパートシステム構築ツールは、ルール、フレームなどの知識表現手段と、それに対する推論機構を具備することが 一般的である [Barstow, et al. 1983]。最近では、プログラミング環境[Barstow, et al. 1984] という用語が定着してきたことからもわかるように、エディタ、デバッガ、ブラウザ、ファイルシステムなどを統合した エキスパートシステム構築ツールが提供されるようになり、エキスパートシステムの構築は 一段と加速している。

一般に エキスパートシステムは、ナレッジエンジニアにより 構築される。ナレッジエンジニアは知識を収集するため、エキスパートにインタビューしたり エキスパートの問題解決事例を分析する。そのため、これまでに論じられている多くの環境は、ナレッジエンジニアのためのものである。

一方「エキスパートは 自分が思っている以上に知っている」といわれるようにエキスパートシステム開発時に、ナレッジエンジニアが 正確かつ完全な知識を収集することは困難である。このことに注目して、エキスパート向けの知識獲得ツール（MOLE[Eshelman, et al. 1986], CSRL[Bylander, et al. 1986] など）も研究されている。

これまでのエキスパート用のツールは、エキスパートの問題解決手法（専門家モデル）を個別にモデル化し、その専門家モデルに基づいて システムが 直接エキスパートにインタビューすることにより 知識を獲得する。そのため、一つのツールは 診断や分類などに応用分野を限定される。

このように、これまでのエキスパートシステムの構築環境は、 KEE [Fikes, et al. 1985] や ART のように汎用性を指向し 知識モデルだけを提供するナレッジエンジニア用と、 ETS [Boose. 1984] や MOLE [Eshelman, et al. 1986] のように専用性を指向し 専門家モデルまでも提供するエキスパート用 とに分けて論じられてきた。そのため、ナレッジエンジニアにより開発された知識ベースを エキスパートが保守することは困難であったし、エキスパートがプロトタイピングした知識ベースを ナレッジエンジニアが拡張することも困難であった。

本論では、エキスパートシステム構築ツールは、種々の関与者の役割に応じて、互いに関連のある環境を統合するべきであると考える。環境を統合するため、知識ベースビューという概念を導入する。基本的なアイデアは、ナレッジエンジニアが開発した知識ベースに対し、知識ビューと呼ぶ一種のメタ知識を与えることにより、エキスパートやエンドユーザが 彼らの抽象レベルで 知識にアクセスできる手段を提供しようというものである。

はじめに、ビジネス分野におけるソフトウェア開発手順の変遷について考察し、計算機部門による定形処理のプログラミングとエンドユーザ部門における非定形処理のプログラミングとを対比する。次に この対比に基づき、エキスパートシステムの構築モデルを提案する。ここでは、一つの知識ベースに対して 次の４種類のインタフェースが必要であると主張する：

（ａ）試行環境、（ｂ）開発環境、（ｃ）利用環境、（ｄ）保守環境

この４つの環境の中では、開発環境がエキスパートシステムを構築する上での基本となる機能を提供すると考える。そこで、開発環境を実現する要素を明らかにした後、開発環境の上に他の３つの環境を統合するために 知識ベースビューを導入する。この知識ベースビューは エキスパートシステムの専門家モデルの再利用も可能とする。

試行環境は、ナレッジエンジニアにより既にシステム化された問題解決と同じ専門家モデルが使える場合、エキスパートが 短時間で 知識ベースを作成できるようにする。エキスパートは、エキスパートシステムとは何であり、何ができるかを知ることができる。つまり、試行環境では、どのような問題解決手法があるかが分かり、その一つを選択したとき、開発環境で参照するよりも抽象度の高い形式で知識を参照できる。

試行環境で取り揃えられる専門家モデルのメニューは、あらゆる問題を解決できるわけでなく、生成される知識ベースは 必ずしも新しい分野で要求される機能を満足できない。その場合、ナレッジエンジニアが 開発環境において 知識モデルを活用して 知識ベースを作成する。

利用環境を実現する知識ベースビューは、いかなる知識の集合（知識セット）により一つの応用システムが構成されるかを定義するメタ知識である。利用者は どの知識セット（あるいは誰の知識セット）を用いて問題を解くかを選択したり、問題の解がどの知識セットに基づくかを知ることができる。一方、この知識ベースビューは、ある知識セットが一つ以上のエキスパートシステムを構成することも可能とする。

保守環境を実現する知識ベースビューは、エキスパートが 自分の責任範囲について 自分の見やすい表現で 知識ベースにアクセスすることを可能とするためのメタ知識である。ある応用で保守のために定義された知識ベースビューは、専門家モデルが同じ別の応用を試行するときにも利用できる。例えば、ある分類型のエキスパートシステムの保守ビューの組ができると、これを試行ビューとして登録することにより、以後 類似した分類型の応用システムのイメージ作りを行う試行環境ができる。

３．２　エキスパートシステム構築における
試行／開発／利用／保守環境の統合の必要性

３．２．１　ビジネス分野におけるソフトウェア開発手法の変遷

ビジネス分野における本格的なシステム開発は、昭和４０年代から始まった。当初のアプリケーションは、計算機（ＥＤＰ：Electronic Data Processing）部門による大量の伝票処理である。対象領域の特徴は、各ユーザが大量のプログラマ、資金を投入して個別に開発しても投資効果のある大量データの定型処理であり、銀行などの勘定処理システム、鉄道などの座席予約システム、製造業の在庫管理システム などが開発された。

昭和５０年代半ばごろからは、従来のＥＤＰ部門によるプログラミングだけでは、バックログ（プログラム開発要求＞プログラム開発能力の時に、生じるシステム開発の積み残し）が生じること、アドホックな処理ができないこと、が問題として表面化し、エンドユーザ（計算機出力の最終利用者で、問題解決能力のある人）によるプログラミングが行なわれるようになった［Martin, 1986］。このフェーズでは、各ユーザが個別に工数をかけてエンドユーザ用のシステムを開発するには 投資効果の面から値しないため、第４世代のエンドユーザ用簡易言語が注目を浴びた。

以上述べたＥＤＰ部門によるプログラミングとエンドユーザによるプログラミングの比較を表３．１に示す。

表３．１　ＥＤＰ部門によるプログラミングと
エンドユーザ部門によるプログラミング

フェーズ	システムエンジニア	エンドユーザ
要求分析	手順を追う 明確な目標仕様をもつ	試行錯誤で行う 当初仕様が不明確
設　計	正確かつ首尾一貫した 仕様	ラフ・スケッチ
実　現	一ヶ月以上が普通	数日、場合によっては 数時間
保　守	ステップバイステップ プログラム ＋ドキュメント	任意の時点 プログラム ＝ドキュメント
問　題	長いﾀｰﾝｱﾗｳﾝﾄﾞﾀｲﾑ	実現ｱﾌﾟﾘｹｰｼｮﾝの範囲が限定

表３．１で 特に注意すべき問題点は 次の通りである。

（１）ＥＤＰ部門のプログラミングでは、システムの要求分析が完全に完了しないとプログラム開発に着手できない。

（２）エンドユーザ部門のプログラミングでは、ＤＢ検索と四則演算から得られる表作成程度のアプリケーションしか開発できない。

従って、従来の技術だけでは、ＥＤＰ部門がシステムを初期インストールして エンドユーザ部門が継続的に保守していく 応用システムの構築は 困難である。しかしながら、企業等ビジネス分野において残されたシステム化対象領域では エンドユーザが保守しなければならないことが多い。エンドユーザ部門による保守対象は問題領域の知識であり、この残された領域には、推論エンジンと知識ベースを分離したエキスパートシステムのアーキテクチュアが使われると考える。

このように考えてきたビジネス分野のソフトウェア開発手法の変遷を表３．２に示す。

表３．２　ビジネス分野におけるソフトウェア開発手法の変遷

時期	分　野	キーとなる技術	背　景	例
1970~ ＥＤＰ	定型処理	ファイル設計 高速処理、高信頼性 COBOL言語	大量投資に みあう効果	銀行オンライン 鉄道座席予約
1980~ DSS/OA	非定型処理	データベース エンドユーザ言語 APL言語	開発要求 ＞開発能力	ﾚﾎﾟｰﾄｼﾞｪﾈﾚｰﾀ スプレッドシート
1985~ ＥＳ	半定型処理	知識ベース 知識表現／ 知識獲得	継続的保守が 特に大切	診断 スケジューリング

EDP : Electronical Data Processing
DSS : Decision Support System
ＯＡ : Office Automation
ＥＳ : Expert System

3．2．2　エキスパートシステム構築の関与者の分析

エキスパートシステムの構築を 従来の定型処理を対象としたＥＤＰシステム（以下では 単にＥＤＰシステムと呼ぶ）の構築と比較する。特に、関与者、作業内容と作業順序に注目する。

従来のＥＤＰシステムの開発では、要求分析が完全にできるから 設計・開発も正しくできる ということが前提となっている。つまり、ソフトウェアの生涯をいくつかのフェーズに分け、それぞれのフェーズを滝（ウォータフォール）とみなして、全体を上流から下流に至るこれらの滝のつらなりとしてとらえ、次のように考えてきた。

（1）上流から下流へ向かって、より大局的な部分作業は より局所的な部分作業に分割される。
（2）各フェーズ間で渡される情報は 上流から下流へと流れるべきであって、下流から上流へと逆流させるべきではない。
（3）開発上の問題点の多くは、上流の軽視によって生じる。

最近「この前提が成立しない分野が多い」と主張され、プロトタイピングの重要性が指摘されている。プロトタイプ・アプローチとは、ある意味で要求分析の完了を後回しにし、試作を繰り返しながら 実用システムを構築するものである。つまり、実用システムを開発する前に 実際に動作可能なシステムを試作することにより、ユーザ要求に対する仕様の妥当性や当該仕様の実現可能性、性能などを早期に評価し、実用システムにフィードバックする方式が プロトタイピングである。

「エキスパートは 自分の思っている以上に知っている」と言われるように ある領域に関する全ての知識を一挙に抽出することは困難である。完全な要求分析ができないまま開発せざるをえないので、開発当初正しかった知識が その後陳腐化したり、新しい知識が必要とされることがある。そのため、プロトタイピングアプローチをとらざるをえない。

しかし、次の理由から ナレッジエンジニアだけがプロトタイピングを行っても、エキスパートシステムを構築することは困難である。

（1）エキスパートはエキスパートシステムとは何であり、何が出来るのかが分からないため、そもそもシステムに対する要求を出すことが困難である。
（2）エンドユーザがどの知識（あるいは誰の知識）を使うかは事前に分からない。
（3）知識自体が変化する可能性が高いが、ナレッジエンジニアが継続的に知識ベースの保守を担当することは難しい。

つまり、全てのシステム開発フェーズにおいて ナレッジエンジニアがシステムの開発／保守を担当しうるかは 疑問である。ナレッジエンジニアの絶対数が増加することは期待できないし、ナレッジエンジニアをおく投資効果がない場合さえあると思われる。

そこでエキスパートシステムの構築を 関与者との関係で次のように考える。

（1）試行

エキスパート自身が既存のエキスパートシステムの専門家モデルを理解し、それに従って自分の分野の知識の一部を入れることにより システムのイメージ作りを行なう。専門家モデルのメニューが揃っており、その中から一つを選択することにより、簡便に短期間で本格的に開発するか否かを評価できるシステムを作れる必要がある。

（2）開発

ナレッジエンジニアが （1）のフェーズで試作されたシステムの不備を分析し、エキスパートにインタビューすることにより、本格的にシステム開発を行なう。可能であれば、試作システムを拡張することにより システムを構築する。必要に応じて専門家モデルを修正したり、場合によっては まったく新規に専門家モデルを作る。マンマシンインターフェイスなどの開発のために、多くのプログラマも投入される。複数のエキスパートの知識を記憶する必要がある場合も考慮しておく。このフェーズの中でもプロトタイピングを行う。

（3）利用

完成したエキスパートシステムを エンドユーザが利用する。場合によっては ある一つの知識が 複数のエキスパートシステムで使用されることもある。エンドユーザはどの知識、あるいは誰の知識を利用して問題を解くか選択できる必要がある。反対に問題の解にたいし、どの知識あるいは誰の知識によるものかを把握することができる。

（4）保守

エキスパートが 運用されているエキスパートシステムの知識を 洗練していく。知識の保守とは、「ハードウェアでいう故障の修理」「ソフトウェアでいうバグの除去」以上の意味をもち、陳腐になった知識の除去、新たな知識の獲得、知識のマクロ化、不足の抽出、矛盾の除去までも意味する。このフェーズでは 開発したナレッジエンジニアの援助は基本的に得られない と考えておく。

3．2．3　環境の構成

エキスパートシステム構築ツールは、先に示した構築モデルに準拠して機能を提供するべきである。そこで それぞれの環境のことを 試行環境、開発環境、利用環境、保守環境と呼ぶ。それぞれの環境に要求される機能を 表3．3にまとめる。

表3．3　システム構築環境への要求

フェーズ	参加者	目的	要求
試行	専門家	アプリケーション イメージを 短時間で固める	標準的な分野依存知識表現 表やツリー形式 知識表現
開発	ﾅﾚｯｼﾞｴﾝｼﾞﾆｱ プログラマ 専門家	実用目的の アプリケーションを 開発する	強力なハイブリッド型 知識表現 知識ベースに対する セキュリティ 動的・静的検証機能
利用	エンドユーザ	アプリケーションを 活用する	ﾕｰｻﾞﾌﾚﾝﾄﾞﾘｲﾝﾀﾌｪｰｽ 従来ｼｽﾃﾑとのｲﾝﾀﾌｪｰｽ
保守	専門家	継続的に アプリケーションを 維持する	ｶｽﾀﾏｲｽﾞ可能なエディタ ｽﾃｯﾌﾟﾜｲｽﾞ知識洗練化、 一貫性チェックなど

エキスパートシステムの開発フェーズにより要求される機能は異なる

次に この4つの環境の関係について考察する。これらの中では、開発環境が基本となる機能を提供する環境であり、他の試行環境、利用環境、保守環境は、一種のユーザビューであると考える。このように考えたときの環境の関係を 図3．1に示す。

ビューとは、一般に 複雑な実態をある側面からみる見方である。設計に関しては 古くは3次元建築物の立面図、側面図などがある。計算機の分野では、データベースのビュー[Date. 1981] [Ullman. 1980] が ある。

データベースビューは、特にリレーショナルデータベースで論じられており、複数の関係をある項目で結合し、あたかも一つの関係であるかのように見せたり、ある項目にたいし、セキュリティなどの関係から エンドユーザにその存在を知らしめないようにするものである。ここで興味深いのは、もとの関係に対する操作言語とビューで定義した関係に対する操作言語を統一すること をねらっている点である。すなわち、もとの関係に対するユーザもビューに対するユーザも 操作に関して 同じ水準の知識をもつことを前提としている。

それに対し、本論の知識ベースビューは 利用者に対する知識の見せ方の抽象レベルをかえることをねらう。すなわち、複雑な実態にたいし、ある側面からの見方を与えるという意味では同じ目的をおき、操作に関しては 利用者のレベルに応じてかわることが自然であると考える。しかし、第7章で論じるように データベースビューを一種の知識ベースビューとして利用できると考えている。

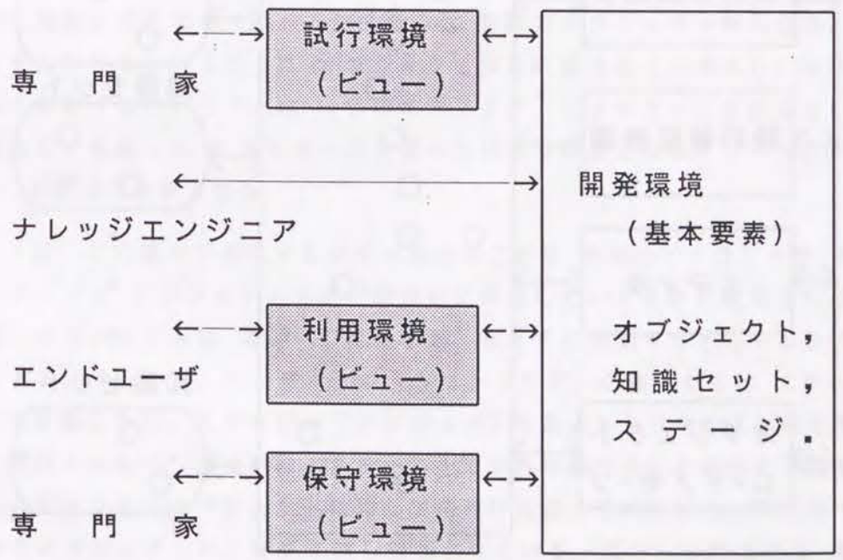


図3．1　システム構築環境と知識ベースビュー

システム構築用基本要素の上に各フェーズに要求される機能を提供するため、ビューと呼ぶ概念を導入する

３．３　エキスパートシステムの開発環境を実現する基本要素

３．３．１　概要

他の環境の基本要素となる開発環境には、ナレッジエンジニアが多数のプログラマとともに参画する。この環境は、知識のデバッグに要するターンアラウンドタイムの削減を目的として設計されるべきである。なぜなら、エキスパートシステムにおける知識の記述には、従来のプログラミングにまして、動的な検証、静的な検証の繰返しが必要とされるからである。

この環境の設計に当たって、知識の基本単位とその組合せ管理体系を如何に設定するかが問題となる。ここでは、記述単位をオブジェクト、保存単位を知識セット、実行単位（動的／静的な検証の対象となるオブジェクト群の範囲を規定する）をステージと名づけた。これらの関係を図３．２に示す。以下それぞれについて、その存在理由と特徴について述べる。

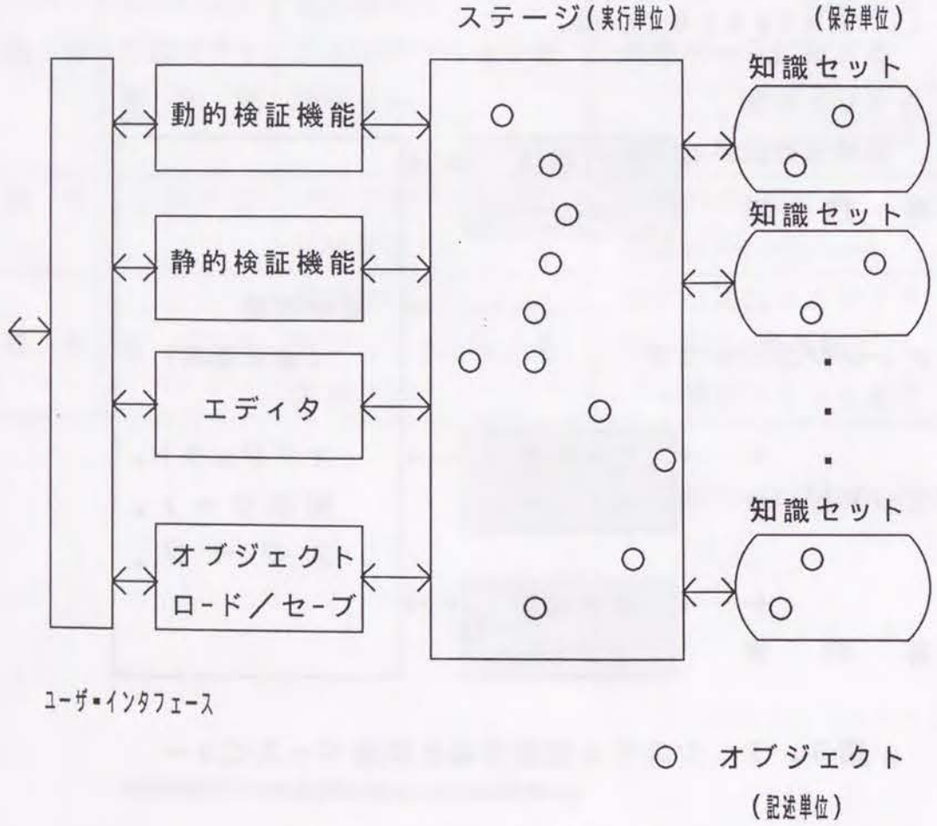


図３．２　開発環境の基本要素

３．３．２　オブジェクト

一般に、知識の記述に関して、次の要件を満たす必要があると考える。

（１）種々のタイプの知識（ルール、デモンなど）を記述できること。

（２）記述単位の間では、互いのモジュラリティが 高いこと。

（１）の要件を満たす知識表現言語は、ハイブリッド型知識表現と呼ばれる。ハイブリッド型知識表現では、オブジェクト（フレーム）がベースとされることが 一般的である[Fikes, et al. 1971] [Ogawa. 1985] [Chusho, et al. 1989]。オブジェクトは表現対象を構造的に記述することが可能であり、あるクラスに属するものの記述を より一般的なクラスの特別なものとして記述するため、（２）の要件を満たすという意味で注目されている。本章では 知識の記述単位を 特にフレーム、ルールなどと区別せず、２．１．３で述べたように 総称してオブジェクトと呼ぶことにする。

ここでいうオブジェクトとは、データ（スロット）と処理（メソッド）が、一体となったものである。オブジェクトを動かすには、該オブジェクトの外界（ユーザ または 別のオブジェクト）から「メッセージを送る」という概念により、メソッドを起動する。送り側のオブジェクトは、受け側のオブジェクトの記述内容をブラックボックスとして見るので モジュラリティが高い。

３．３．３　ステージ

本論で対象とするエキスパートシステムは 複数のオブジェクトからなる。エキスパートシステムを作るということは オブジェクト群を定義することであり、エキスパートシステムを起動するということは 一つのオブジェクトにメッセージを送るということである。従って、定義したり メッセージを送ったりする対象となるオブジェクトの集合に対し、一つの概念が必要となる。

ここでは、この集合が存在する場所の概念のことを 芝居のアナロジーで ステージと呼ぶ。ステージは オブジェクトを試行錯誤的に作成していくときに使用される作業エリアである。オブジェクトは ステージにあるか、後述する知識セットにあるかのいずれかであるとする。すなわち、ユーザ（ナレッジエンジニア）の観点からは ステージにオブジェクトを移動したり、ステージからオブジェクトを除去したりすることができる。オブジェクト間のメッセージ送受信は、ステージ上でのみ可能であるとみなす。編集、検証についてもステージ上のオブジェクトに対してのみ行える。それに対し、次に述べる知識セットの中のオブジェクトは、変更に対し保護されている。ステージの上では オブジェクト名はユニークでなければならない。

ステージという概念は、次の特徴をもつ。

（１）知識のデバッグは試行錯誤を伴うので 知識の修正の都度 元の知識に置換するのではなく、作業的に知識の現況を保つことができる。

（2）推論途上の状態を保存したまま セッションを終了し、次回 その状態から再開することが可能となる。

（3）応用システムとして意味のある全てのオブジェクトをステージにロードしなくても 一部の知識を用意するだけでデバッグが可能である。

（4）開発担当者毎に一つ与えられる。

3．3．4　知識セット

ステージ上にないオブジェクトの保管場所の概念を 知識ベースと呼ぶ。知識ベースを更に、知識セットと呼ぶ概念に分割する。知識セットの概念の導入の目的は 次のとおりである。

（1）複数オブジェクトのマクロ化機能である。セット単位で知識ベースからステージにローディングできる。

（2）オブジェクト名称をユニークに保つ単位である。そのため、セットが異なれば、同じ名称のオブジェクトが存在してもよい。

（3）オブジェクトの所有者（知識源）を明らかにする。そのため、セットは、誰もが参照できる「公用」、個人個人に与えられる「私用」という属性をもつ。

知識セットの中のオブジェクトは 直接修正されたり 検証されたりすることは無く、その意味で保護されている。

3．3．5　オブジェクト指向のユーザインターフェイス

オブジェクトに対し、処理する手順として 次の２通りが考えられる。

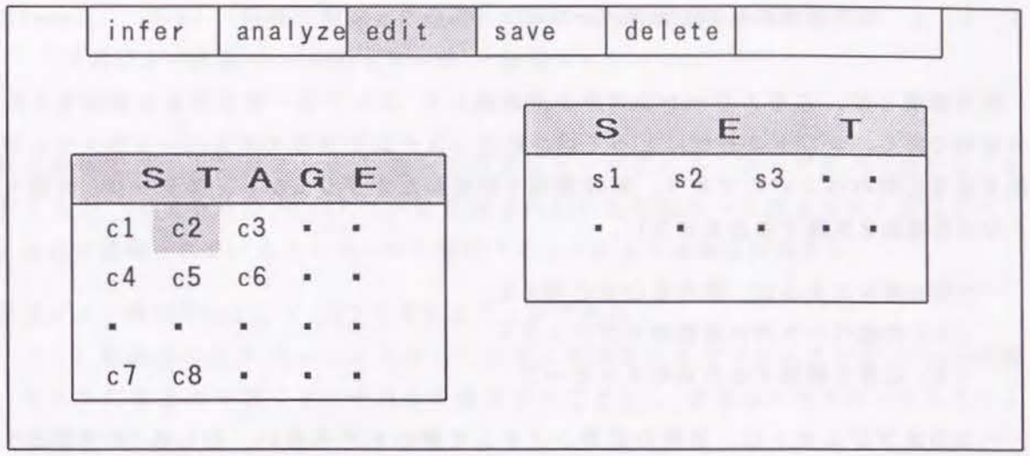
（1）はじめに処理方法（例えば、編集）を指定した後、その対象とするオブジェクトを指定する。

（2）はじめに対象とするものを指定した後、それに対する処理法を指定する。

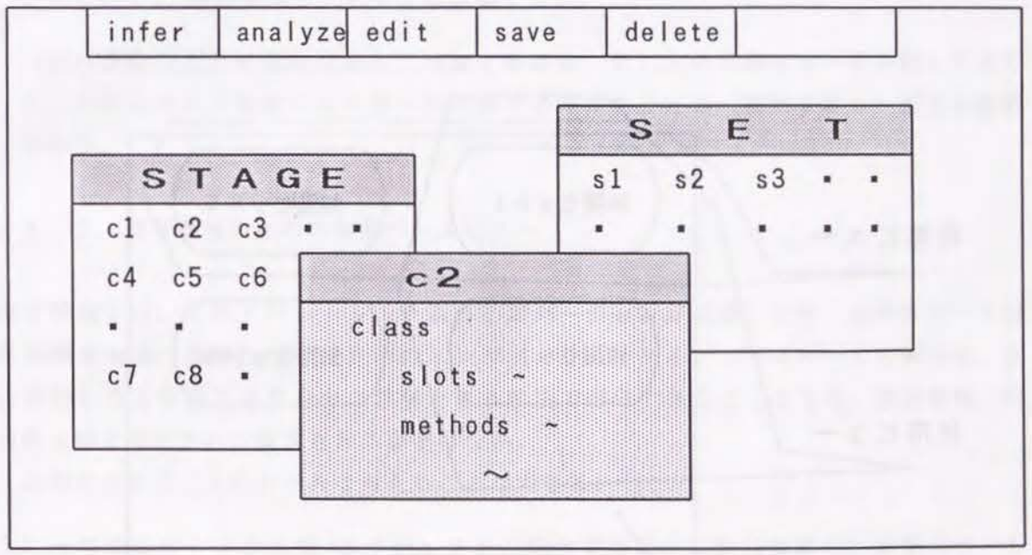
これらの操作方法の違いは 一つの手順の後の手順の移行時に顕著に表れる。（1）の手順では、例えば あるオブジェクトを編集した後、システムは次に何を編集するかという状態になる と考えられる。それに対し（2）の手順では、そのオブジェクトに対し、次に何をするか という状態になると考えるべきである。

我々は、知識のデバッグ時には 続けて種々のオブジェクトを編集するのではなく、むしろ、編集中のオブジェクトを解析してみたり、解析中のオブジェクトの内容を見たりというオブジェクト中心の操作が多い と考える。

従って、開発環境の操作方法は（2）の手順にするべきだ と考える。この手順による操作例を図３．３に示す。



（1）オブジェクト（c2）と処理法（edit）を選択。



（2）オブジェクト（c2）を編集するウィンドウが開かれる。

図３．３　オブジェクト指向のユーザインタフェース

３．４　知識ベースビューによる試行／利用／保守／再試行環境の提案

３．４．１　利用環境のための知識ベースビュー

利用環境とは、エキスパートシステムが完成して エンドユーザがそれを操作するための環境である。エンドユーザにとっては、応用システムであるエキスパートシステムの操作方法さえ知ればよいのであり、開発環境で設定したオブジェクト、ステージ、知識セットなどの概念を意識する必要はない。

一つの応用システムは、次からなると考える。

（１）知識ベース内の複数のオブジェクト

（２）応用を開始するためのメッセージ

一つのオブジェクトは、複数の応用システムで使われても良い。むしろ オブジェクトのモジュラリティが高いならば 積極的に複数の応用システムで使うべきである。このように考えると、応用システムは 知識ベースに対する論理的な視点として考えられる。ここでは、この視点を利用ビューと呼ぶ。利用ビューの位置付けを図３．４に示す。

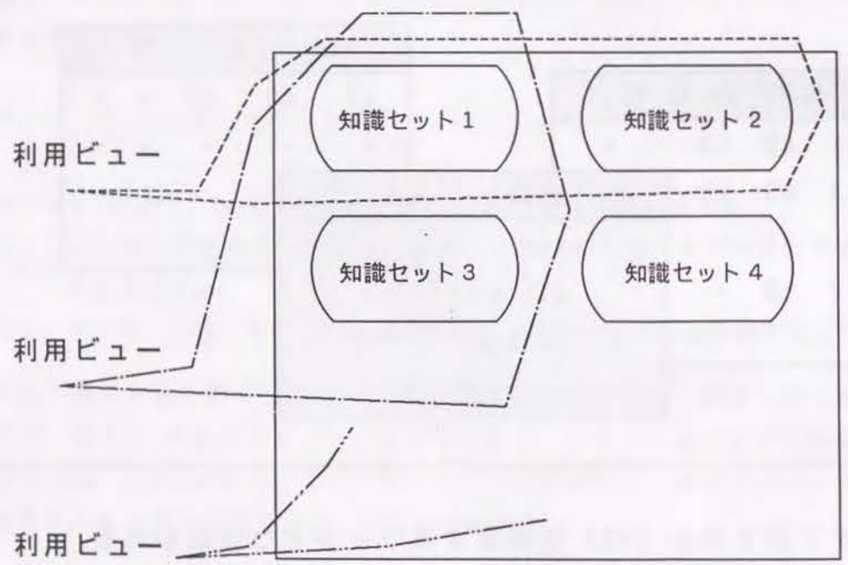


図３．４　利用環境のための知識ベースビュー

ここで 一つの応用には多数のオブジェクトが利用されることが多いので 利用ビューの定義は知識セットを単位としている。つまり、利用ビューは形式的にはＡＮ記法［Shimauchi, 1972］（概要を付録３に記載）で次のように定義される。

<利用ビュー定義>===<利用ビュー名> <開始メッセージ>,
<知識セット名>{, }。。。

利用環境では、エンドユーザは 複数の利用ビューが見える。ユーザが一つのビューを選択すると、システムは そのビューに定義されている知識セット群をステージにロードし、同じく定義されているメッセージを発行することにより推論を開始する。

利用ビューの使用法として 以下を考えることができる。

（１）複数のエキスパートシステムで共通に使用されるオブジェクトを 一つの知識セットにまとめておくと、それらを複写することなく 複数のエキスパートシステムで共用できる。

（２）応用問題を解決する上で複数の観点がある場合（例えば、査定支援システムで、厳しく評価する知識と緩く評価する知識があるなど）それらを分けて知識セットにまとめておくと 評価知識の入れ替えが容易である。

（３）多数の人の知識を収集して推論する場合、それらの知識セットを分割しておくと、いかなる人の知識により得られた解であるかが分かる（説明できる）応用を構築できる。

３．４．２　保守環境のための知識ベースビュー

保守環境とは、エキスパートシステムの初期バージョンが完成した後、エキスパートが自ら知識を充実、洗練していくためのインタフェースである。エキスパートの要求は、自分の関知しうる知識だけを自分の理解できる構造で保守できることであり、開発環境、知識表現仕様で設定された概念を知る必要はない。

この限定は次の二つのレベルで考えることができる。

（１）一つの応用システムは オブジェクトの集合である。しかしながら、エキスパートのもつ知識を記述するオブジェクトは そのサブセットである。オブジェクトには、データを表示するためのものなど応用システムをインプリメントする都合上のものがある。エキスパートは、この種のオブジェクトの内容には興味を示さないし、その存在すら知りたくない。

（２）オブジェクトの記述仕様は 応用領域を限定しないために強力な記述力をもつように設定されるべきであるが、その反面 分野のエキスパートが その言語で知識を記述するのは困難である。しかしながら、多くのエキスパートシステムでは、一度システムがインプリメントされると、保守の対象となる知識には構造がある。

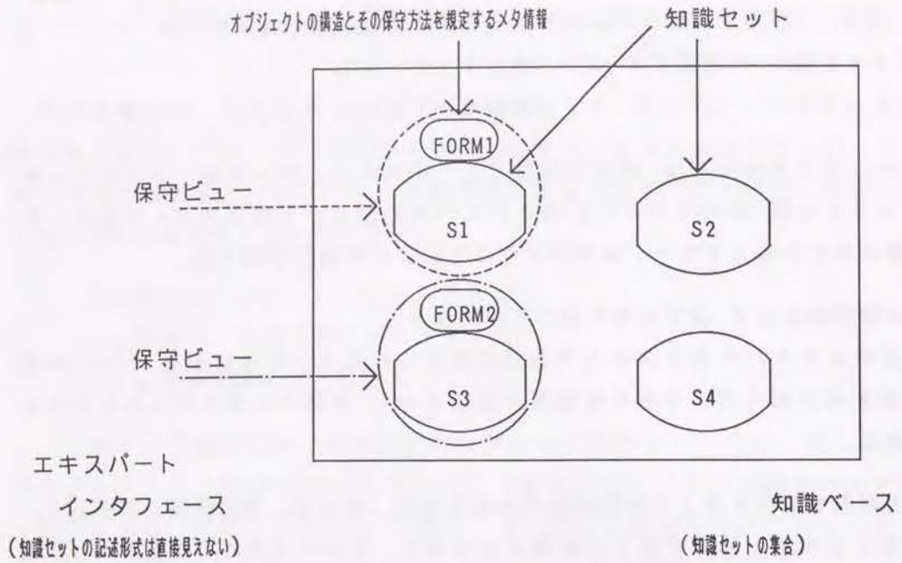


図３．５　保守環境のための知識ベースビュー

- 保守対象となる知識セットは知識ベースの一部である
- 保守対象の知識セットには、陽に記述できる構造がある

これより 保守環境において、一つの応用システムは 次の概念からなると考える。

（１）知識ベース内のオブジェクトのサブセット

（２）オブジェクトの構造とその保守方法を規定するメタ情報

ここでは、この概念の組を保守ビューと呼ぶ。特に、後者のメタ情報のことをＦＯＲＭと呼ぶ [Tsuji, et al. 1987b]。保守環境において、エキスパートは 保守ビューが見える。保守ビューの位置付けを 図３．５に示す。保守ビューは形式的には次のように定義されるがこの詳細については 次章以降で論じる。

<保守ビュー定義>===<保守ビュー名> <知識セット名>
<メタ情報>

エキスパートが 一つの保守ビューを選択すると、システムは そのビューに定義されているオブジェクトを前述のステージにローディングする。次に、エキスパートが その中の一つのオブジェクトを選択すると、システムは 同じくビューに定義されているＦＯＲＭに従って エキスパートに対し オブジェクトの保守を誘導する。そのため、ＦＯＲＭには、エキスパートとの対話法を含む。

３．４．３　試行環境，再試行環境のための知識ベースビュー

保守環境を実現するために用いられるＦＯＲＭは 必ずしも応用システム毎に異なるとは限らない。例えば、分類型のエキスパートシステムとして応用システムの目的を設定することにより、ＦＯＲＭの組ができ、複数のエキスパートシステムに利用できる場合がありうる。このような組をツールが標準的に具備すれば、エキスパートが自ら応用システムのイメージ作りを簡単に行う試行環境の構築が可能となる。

試行環境の位置付けを図３．６に示す。すなわち、当初、試行環境はなく、開発環境（図３．６－１）で知識ベースが開発されていく。知識ベースが構築されると利用ビューにより利用環境（図３．６－２）が、保守ビューにより保守環境（図３．６－３）が構築される。

保守ビューがとり揃ってくると、それらを集めて試行環境（図３．６－４）を構築する。この環境で別の応用システムのプロトタイプを行う。ある場合には試行環境でできたエキスパートシステムをそのまま利用することができる（図３．６－５’、６’）。ある場合には、開発環境に戻って、知識ベースを拡張・修正する（図３．６－５）。拡張した知識ベースに対して利用ビュー、保守ビューを設けることにより、新しいエキスパートシステムの利用環境、保守環境ができる（図３．６－６，７）。このとき、新たな保守ビューを試行環境に追加することにより、試行環境のレパートリを増やすことが可能（図３．６－４）と考えている。

このエキスパートシステム構築のリサイクルの例については 第５章で論じる。

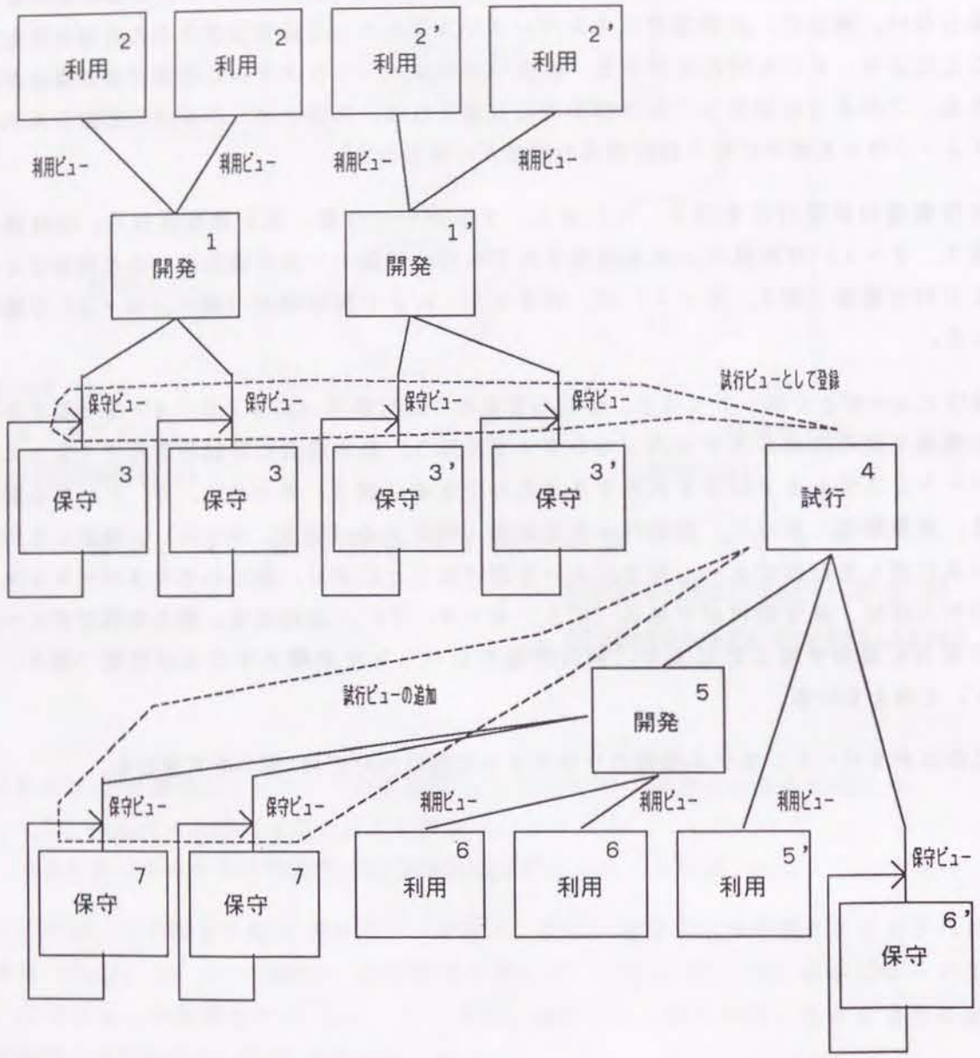


図３．６　エキスパートシステム構築のリサイクルモデル

図３．６の補足説明

開発環境（１）で知識ベースを開発。

開発した知識ベースに利用ビューにより利用環境（２）、保守ビューにより保守環境（３）を構築。

保守ビューを収集して試行環境（４）を構築。

試行環境（４）で別の応用システムのプロトタイピング。その結果：

①試行環境でできたエキスパートシステムをそのまま利用可能（５’、６’）

②開発環境に戻って、知識ベースを拡張・修正（５）。

拡張した知識ベースに対して利用ビュー、保守ビューを設け、新しいエキスパートシステムの利用環境、保守環境を構築（６，７）。

新たな保守ビューを試行環境に追加（４）。

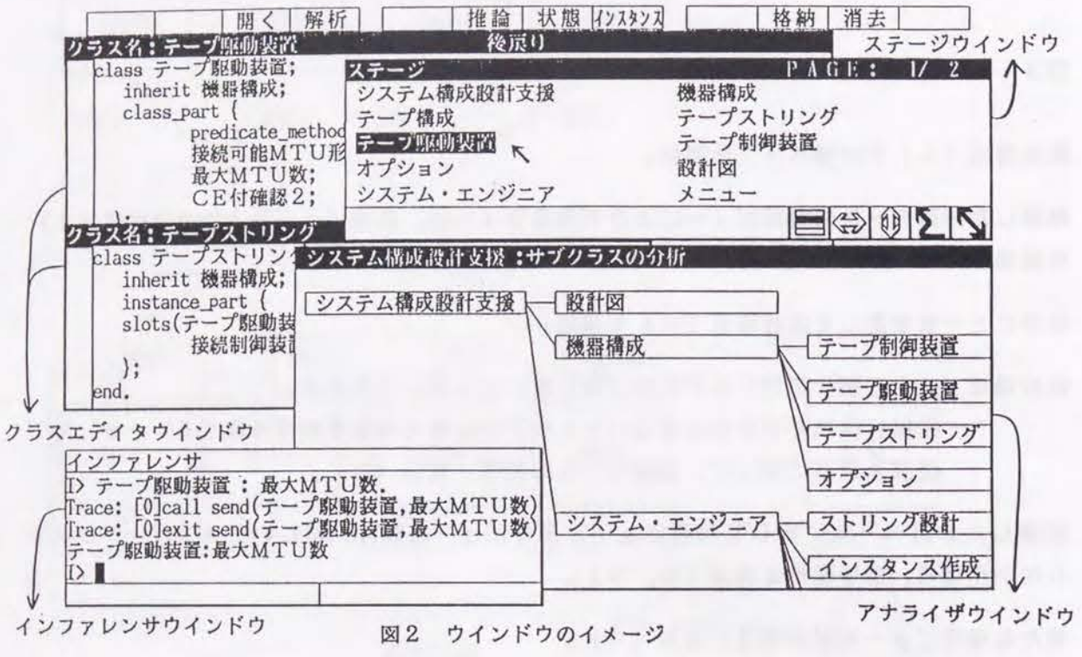


図2　ウインドウのイメージ

図３．６　開発環境におけるユーザインタフェースの画面例

３．５　まとめ

本章では、計算機部門によるプログラミングとエンドユーザ部門によるプログラミングとを比較すると共に ソフトウェア開発手順の変遷を分析することにより、ビジネス分野におけるエキスパートシステムの構築環境について論じた。分析結果として、エキスパートシステム構築ツールには、４つの環境、すなわち試行環境、開発環境、利用環境、保守環境が必要であることを導いた。さらに、それらの環境は 互いに関連しているべきとの主張のもとに、知識ベースビュー（利用ビュー、保守ビュー）、オブジェクト、ステージ、知識セットという概念を導入した。

導入した知識ベースビューの実現性を確認するため、本章の３．３，３．４で提案した諸概念を具備したエキスパートシステム構築ツールＥＳ／Ｘ90 (Expert System Tool for 90's) を開発した。稼働ハードウェアはＣＷＳ２０５０ワークステーションであり、ＵＮＩＸの下で動く。記述言語はＣとＰＲＯＬＯＧである。

他の環境に対しベースと位置付けられる開発環境では、ナレッジエンジニアからは３．３節で示したように 自分のステージと自分がアクセスできる知識セット群が見える。ステージ上には 複数のオブジェクトが編集・検証・推論対象となっている。ステージ上の一つのオブジェクトを選択すると、それに対し処理できるコマンド（推論・解析・編集・保存・削除など）が 画面の上部に表示される。

一つのオブジェクトを編集するには ウィンドウを開く。エディタは 知識表現に沿った構造エディタである。編集ウィンドウは 一つのオブジェクトを表しているので、それに対するコマンドは ステージウィンドウで一つのオブジェクトを選択したときに表示されるものと同様である。

ステージ上のオブジェクトの静的な関係を図示することができる。図の中のオブジェクトも また 選択でき、それに対するコマンドは 先に述べたものと同様である。図３．６は、ＥＳ／Ｘ９０の開発環境のユーザインタフェースの例である。図から判るように ユーザはステージを中心として その上にあるオブジェクトの一覧を鳥かんしたり、特定のオブジェクトを選択することにより 修正することができる。このように開発環境は知識ベースを一から構築するための強力な機能を有している。

一方、この知識ベースに対して 利用ビューを定義することにより利用環境ができる。利用環境では エンドユーザに対し はじめに ビュー名一覧が表示される。エンドユーザは、その一つを選択することにより エキスパートシステムの実行を開始することができる。このとき、エンドユーザは オブジェクト、知識セット、ステージなどの概念を意識する必要はない。

ＥＳ／Ｘ９０の知識表現については、別途 報告されている [Chusho, et al. 1989]。この知識表現は 汎用性を指向しており、記述力が高い反面、エキスパートによる直接定義は 困難である。汎用知識表現で記述された知識ベースに対して 保守ビュー（この

記述方法の詳細は 第４章で記述）を定義すると エキスパートによる知識ベースの保守が可能となる。この例については 第５章で述べるが、知識ベースを保守するエキスパートは、開発環境の提供する強力な言語仕様を知る必要はない。

一度開発した保守ビューを試行環境に登録すると、類似応用システムのプロトタイピングに使える。試行環境で作成したエキスパートシステムが 新しい領域に不向きの場合、開発環境でその知識ベースを拡張／修正することができる。拡張／修正した知識ベースに対して 利用ビュー、保守ビューを定義することにより 新しいエキスパートシステムを実用化することができる。このように知識ベースビューは エキスパートシステム構築時に そのアーキテクチュアをリサイクルすることを可能とする。

以上まとめたように、知識ベースビューは 開発環境のうえに他の３つの環境を統合するためのものである。この知識ベースビューの導入により、ナレッジエンジニアが開発した知識ベースをエキスパートが保守すること、一度構築されたエキスパートシステムのアーキテクチュアを類似のエキスパートシステムの試行に再利用することが可能となる。このため、提案した環境は、ビジネス分野のアプリケーションの構築面から見て新しい領域を開拓すると考える。

第４章以降では、本章で提案した諸概念の具体的なインプリメント法と応用について論じ、提案の有効性を実証していく。

第４章　知識ベースビューによる知識ベースの静的保守方式の提案

４．１　まえがき

本章では、第３章で導入した知識ベースビューの具体化の一例として 知識ベースを静的に保守する方式について述べる。ここで、静的という意味は 推論過程と独立であることを意味し、動的に保守する方式については 別途 第６章で論じる。

目的は、一つの応用分野における問題の解法（専門家モデル）が 分析された時点で、エキスパートが利用できる知識ベースの保守ツールを容易に実現することにある。基本的なアイデアは、ナレッジエンジニアに汎用の知識表現（ルール、フレームなど）で記述された知識ベースの構造と属性を認識させ、それをメタ知識として定義させることにより、エキスパートによる知識ベースの保守を可能にすることである。

そのために、知識ベースについて 以下の仮定をおいた：

（１）一度知識ベースがインストールされると、基本的に変更されない部分と継続的に変更される部分がある。

（２）後者の継続的に変更される部分にも 陽に定義できる構造がある。

これらの仮定のもとで、ＲＡＮＧＥ，ＦＯＲＭと呼ぶメタ知識と ＴＡＩＬＯＲと呼ぶそのインタプリタを提案する。

ＲＡＮＧＥは 上記（１）の仮定から導入されるものであり、ナレッジエンジニアがアクセスする知識ベースの範囲とエキスパートがアクセスする知識ベースの範囲を区分する。一方、ＦＯＲＭは 上記（２）の仮定から導入されるものであり、パターン、ファセットおよびメソッドと呼ぶ３種の概念からなる。パターンは 知識ベースの構造を表し、ファセットは 知識ベースの属性を表し、メソッドは 知識ベースの変更手法を表す。

ＴＡＩＬＯＲは これらのＲＡＮＧＥとＦＯＲＭを解釈し 知識ベースビューとして次の役割を果たす：

（１）知識ベースのいかなる部分を保守できるかを示す。

（２）保守のために穴埋め方式（fill-in-the-blank）のユーザインタフェースを提示する。

（３）重複した知識や矛盾した知識を検出し 警告を発する。

（４）エキスパート用の知識表現をナレッジエンジニア あるいは 推論エンジン用の知識表現に変換する。

計算機システム構成設計支援エキスパートシステムにおける記述例についても述べる。なお、本章で提案する方式の検証は 第５章にて 詳細に行う。提案するメタ知識のインタプリタＴＡＩＬＯＲは エキスパートが関与する知識だけを 自然語形式で提示する 広義のエディタとみなすことができる。

４．２　知識ベースの構造に関する分析

「エキスパートは 自分が思っている以上に知っている」といわれるようにエキスパートシステム開発時に、エキスパートから 正確かつ完全な知識を抽出することは 困難である。場合によっては、開発時点に正しかった知識が陳腐化したり、時々刻々と新しい知識が得られることもある。このように考えると、エキスパートシステムが継続的に使われるか否かは、そのシステムの知識ベースの保守性に依存しているといっても過言ではない。

従来提供されてきたエキスパートシステム構築ツールは、汎用の知識表現法を提供する[Fikes, et al. 1985] [Barstow, et al. 1983]。そのため どのような問題領域に適用できるのかが明確でなく、知識獲得が困難である という指摘がなされている[McDermott. 1986] [Clancey. 1984]。この問題意識に基づき、エキスパートの問題の解法を分析して知識の役割を明らかにすることにより 知識獲得を容易にする試みがなされている。

例えば、ＭＯＬＥ [Eshelman, et al. 1986] は 診断型エキスパートシステムのための知識獲得ツールであり、ＣＳＲＬ [Bylander, et al. 1983] は 分類型エキスパートシステムのための知識表現言語である。ＳＥＡＲ [van de Brug, et al. 1986] は 計算機システム構成エキスパートシステムの知識獲得ツールである。

知識ベースの保守を強調したツールとしては、TEIRESIAS [Davis, et al. 1982] が有名である。TEIRESIAS は、知識の構造に関する知識をメタ知識と呼び、これを用いて知識ベースの保守を実現している。このシステムは 診断型のエキスパートシステムであるＭＹＣＩＮのフロントエンドとしてインプリメントされている。

エキスパートによる問題の解法を分析して知識獲得ツールを実現することは 知識ベースの保守を容易にする有効な方法である。しかし、新しい応用分野が見つかる都度 新しいツールを労力をかけて開発することは合理的ではない。

本章で論じる保守方式は、３．２．２で述べた（４）の保守環境を実現するためのものである。一般に 知識ベースを保守するためには、知識ベースを編集するツールと その正当性を検証するツールが必要である。検証のためには、ナレッジエンジニアが例題をシミュレーションしその推論過程をトレースする知識ベースを記述することを仮定し、以下では 編集に考察の焦点をおく。

我々は、いくつかのエキスパートシステムを汎用知識表現言語を用いて開発した経験[Tsuji, et al. 1987c] [Tsuji, et al. 1987d]から以下の仮説をおいた。

（ａ）開発環境で構築された知識ベースの知識には、一度定義すると変更されない部分と、継続的に保守される部分がある。

例えば、ルールが保守対象となることが一般的であるが、推論方式、競合解消戦略などは 応用によっては 変えられることはないことがある。また、知識の中にデータをスクリーンに表示するための手続きなどを含む場合が多いが、この種の知識もエキスパートにより変更される必要はない。

（ｂ）継続的に保守される部分の知識には、当初開発する上では必ずしも気付く必要はないが、構造や属性がある。

例えば、あるルール群の中の条件部、行動部にある決まった形式が見られることが多い。

これらの仮定のもとで、知識ベースの保守に関して メタ知識（ＲＡＮＧＥ，ＦＯＲＭ）が存在すると考える。本章の開発目標を図４．１に示す。ＴＡＩＬＯＲは、ＦＯＲＭを参照することにより推論エンジンのインタフェースとなる汎用知識表現とエキスパートのインタフェースとなる分野別知識表現との間の変換ツールであり、エキスパートに対し知識ベースビューを提供する。

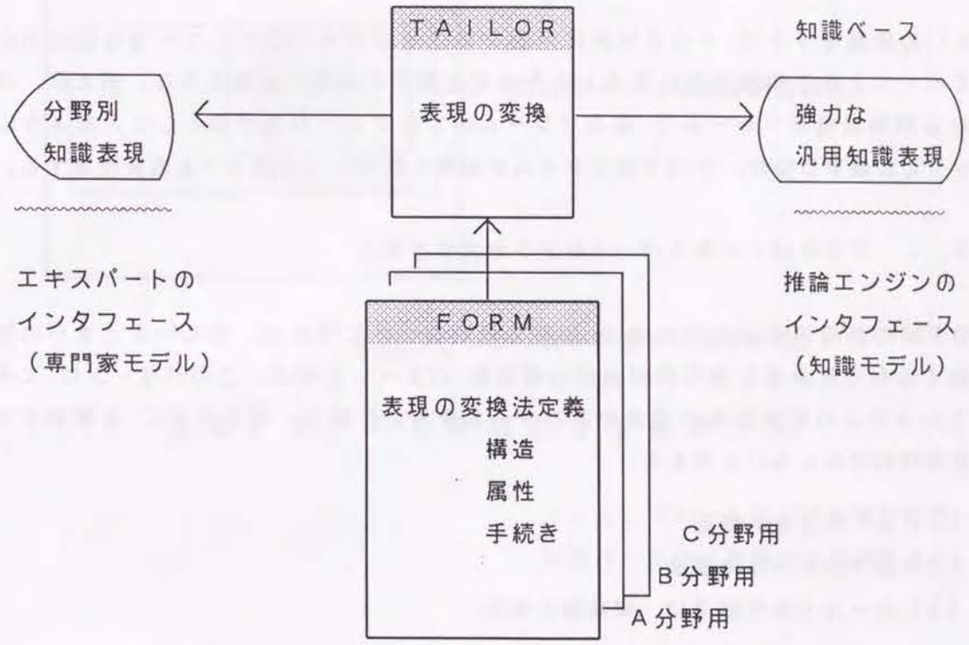


図４．１　ＴＡＩＬＯＲの目標

４．３　知識ベースビューを実現するメタ知識表現とそのインタプリタ

４．３．１　ＲＡＮＧＥ：エキスパートがアクセスする対象

保守環境において、エキスパートは 知識ベース全体を知る必要がない。変更される可能性のある知識にたいしてのみアクセスできればよい。この規定をＲＡＮＧＥと呼ぶ概念で実現する。

ＲＡＮＧＥの定義情報は 次からなる。

（１）ＦＯＲＭ

（２）保守対象知識が記憶されている知識セット

（３）ファセットを規定する知識が記憶されている知識セット

（３）の知識セットは ＦＯＲＭ内に定義するファセット（これについては後述する）としてベースとなる知識表現に依存したものを定義する場合に必要となる。例えば、ベースとなる知識表現がフレームで あるフレームの下位フレーム名が値として入るようにファセットを定義する場合、その下位フレームが記憶されている知識セット名を指定する。

４．３．２　ＦＯＲＭ：エキスパートがアクセスする見方

ＦＯＲＭの表現法を決めるために 知識の構造について考える。ある一まとまりの知識に存在する保守対象部と保守非対象部の構造を パターンと呼ぶ。このパターンは エキスパートシステムの知識表現の代表であるルールを考えた場合、次のように ３階層まで定義できなければならないと考える。

（１）ルール全体の構造

（２）個々のルールの構造

（３）ルールの条件部又は、帰結部の構造

３階層のパターンを 上位から親パターン（pattern）、子パターン（cpattern）、孫パターン（gpattern）と呼び、保守対象部の最小単位のことをブランク（blank）と呼ぶ。これを図４．２に示す。

各パターンは その下位のパターン、ブランクと固定文字列により 構造が定まる。パターン定義を見るだけで知識の構造がわかるように表現法を設計すると、親パターン定義はＡＮ記法［Shimauchi, 1972］（概要を付録３に記載）で次のように 表現できる。ここで｜は代替案定義を示し、。。。は繰返し定義が可能であることを示す。

<親パターン定義>=== (<文字列>|<ブランク>|<子パターン名>)。。。

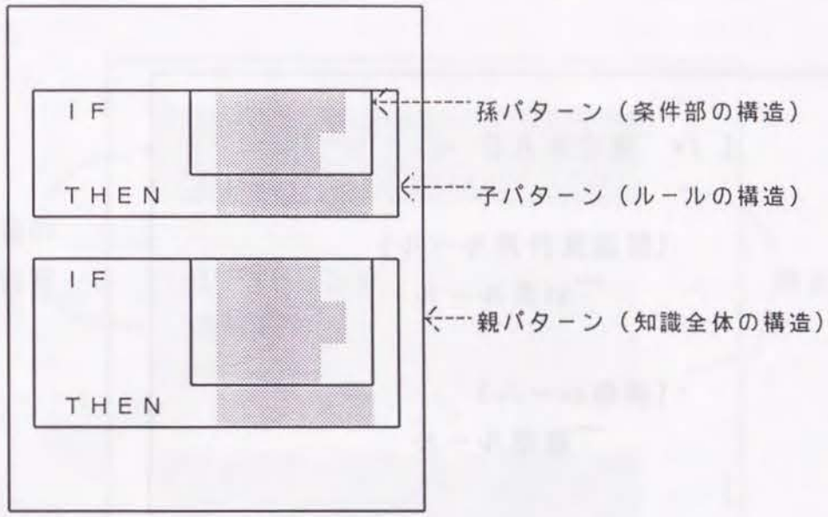


図４．２　ＦＯＲＭの３階層モデル

ルール、フレームなどの知識を定義した
知識ベースには再帰的な構造がある。

２種類のルール群があり、記述の冒頭に日付が /* で囲まれてコメントアウトされている知識に対する親パターン定義の例を 図４．３に示す。ここで <文字列> は、¯を含まない文字の列とし、¯ではじまる文字の列（¯保守年月日）を <ブランク> 、¯¯ではじまる文字の列（¯¯計算ルール、¯¯連想ルール）を <子パターン名> とする。

```
{ /* ¯保守年月日 */

    {関連度計算ルール}
        ¯¯計算ルール

    {連想ルール}
        ¯¯連想ルール
}
```

図４．３　親パターンの記述例

２つのルール群からなる知識に対する定義例

不当な値がブランクに混入しないように、ブランクに対する属性を定義できるようにする。この属性をファセットと呼ぶ。ＦＯＲＭのモジュール性を高め、知識ベースの値を更新する手段の所在を明らかにするため、ブランクに値を挿入する手続きをＦＯＲＭの中に定義する。この手続きをソーイングメソッド（sewing-method）と呼ぶ。以上の考え方を仕様として表現した結果を示す。

```
<ＦＯＲＭ定義>  ===  form <フォーム名>;
                       pattern{<親パターン定義>}
                       [cpattern{<子パターン定義>(,)。。。}
                       [gpattern{<孫パターン定義>(,)。。。}]]
                       [facet{<ファセット定義>(,)。。。}]
                       sewing-methods{<ソーイングメソッド定義>。。。}
```


ここで アンダラインのある語は パーサのための予約語であり、他の文字はＡＮ記法のメタ記号（付録３参照。例えば、［］内は省略可能であることを示す）である。ＦＯＲＭは、pattern, cpattern, gpattern, facet をまとめてスロットとみなせば、ひとつのオブジェクト［Suzuki. 1985］とみることができる。すなわち、sewing-methods を使わずにＦＯＲＭの外界からパターンの中の変数の最小単位であるブランクに直接アクセスされることはない。

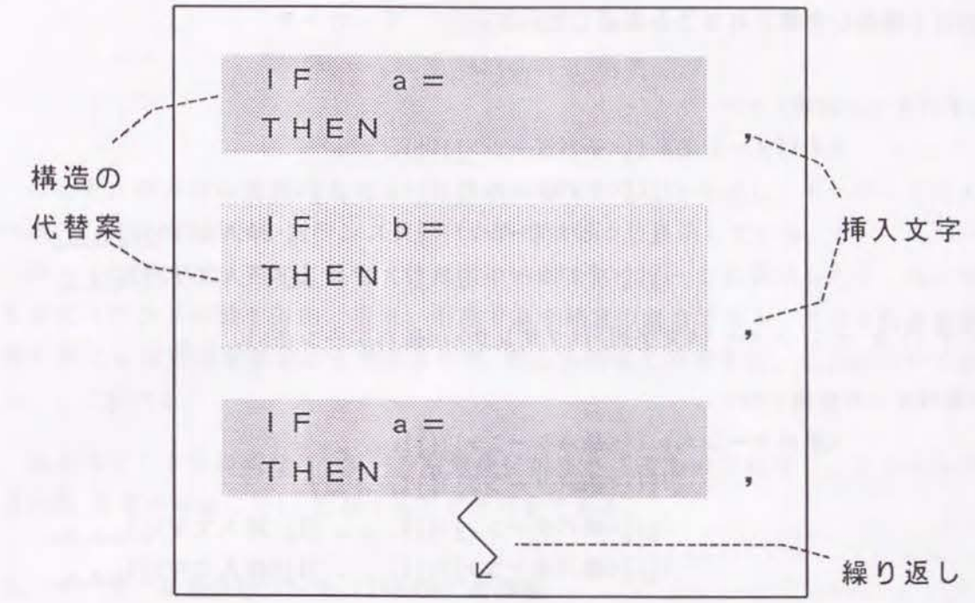


図４．４　ＦＯＲＭの記述力への要求

ＦＯＲＭの記述力を強化するため、図４．４に要求を示すように子パターン及び孫パターンの構造には 次の指定ができる必要がある。

(1)複数の構造の代替案

ある知識の構造は、Ａの型か Ｂの型かどちらかであるという表現を可能とする。

(2)省略可否

知識ベースのある部分に関し、値が設定されることもあれば設定されないこともあるということを表現する。

(3)繰返し

ルールなど ある構造をもった部分が 任意個続いて定義されることを表現する。

(4)繰り返す場合の挿入文字

(3)で 任意個続いて定義可能 とした場合に、互いの間を区切る文字がある場合 それを定義する。

この要求を満たすための子パターン定義、孫パターン定義を次に示す。（ＦＯＲＭ定義の形式定義にもＡＮ記法を用いており、例えば、付録３に示すように。。。はこの左の要素が１回以上繰返し定義されることを示している）

```
<子パターン定義> ===
        <子パターン名> ((<子パターン>))(|)。。。
                |[((<子パターン>))(|)。。。]
                |(((<子パターン>))(|)。。。)[{<挿入文字>}]。。。
                |[((<子パターン>))(|)。。。][{<挿入文字>}]。。。

<子パターン> === (<文字列>|<ブランク>|<孫パターン名>)。。。

<孫パターン定義> ===
        <孫パターン名> ((<孫パターン>))(|)。。。
                |[((<孫パターン>))(|)。。。]
                |(((<孫パターン>))(|)。。。)[{<挿入文字>}]。。。
                |[((<孫パターン>))(|)。。。][{<挿入文字>}]。。。

<孫パターン> === (<文字列>|<ブランク>)。。。
```

図４．３で例示した~~計算ルールに対する子パターン定義の例を次に示す。ここで、~~~ではじまる部分（~~~キーワード、~~~関連度加算）が <孫パターン名> であり、ルール番号がブランクである。

```
~~計算ルール    (関連度計算ルール~ルール番号
                    if  ~~~キーワード
                    then ~~~関連度加算)。。。
```

同様に、この子パターン定義内の孫パターンを詳細化した定義の例を次に示す。孫パターン定義では任意の文字列と~で先行するブランクからなっている。

```
~~~キーワード   (  ~キーワード の @関連を を ?x とし
                        @関連度 が 0 より大きい  )

~~~関連度加算   (  send ~教訓 加算(~関連度、?x)  )
```


ファセットの種類は、ブランクに入る値の型、値の候補の他、ベースとなる知識表現の仕様に依存して決まる。例えば、あるブランクに知識ベースに別途記憶されているフレームの下位のフレーム名が入る場合などに定義すると有効である。ファセット定義の例を示す。

```
~関連度      ( type ( real ), range ( 0, 1) )
~キーワード  ( instance ( keyword ) )
~教訓        ( instance ( 教訓 ) )
```

ここで、関連度は実数の方で０～１の間の値をとることを示し、キーワードに入る値はkeyword フレームのインスタンスフレーム名であることを示している。

ソーイングメソッドは、保守する知識の内容をスクリーンに表示したり、ユーザから値を得てブランクの値を追加、変更、削除する手続きの集合である。この手続きを個別に開発することは労力を要すことであるので 組込手続きを用意する。これについては、４．３．３で述べる。

以上のＦＯＲＭ表現法は ルール型知識に限定されるものではなく、フレームや述語などの他 任意の知識に対し 定義することが可能である。

４．３．３　知識保守ツール（TAILOR）の機能

ＴＡＩＬＯＲは 一つの知識ベースビューを与えられると、次のように動作する。

[step1]　ＦＯＲＭを解析する

[step2]　保守対象となる知識を、step1 で認識したＦＯＲＭで 固定文字列部と変数部に分解する。知識とＦＯＲＭの間に矛盾があり 分解できなかった場合は停止する。

[step3]　ＦＯＲＭに定義されているソーイングメソッドの一覧を 表示し、選ばれたものを処理する。この処理過程で ブランクに値が設定される。

[step4]　知識を生成して 知識ベースに出力する。但し、ブランクのなかに未設定のものがあれば その旨示す。

以上述べた動作を実現するＴＡＩＬＯＲの構成を 図４．５に示す５つのコンポーネントからなる。

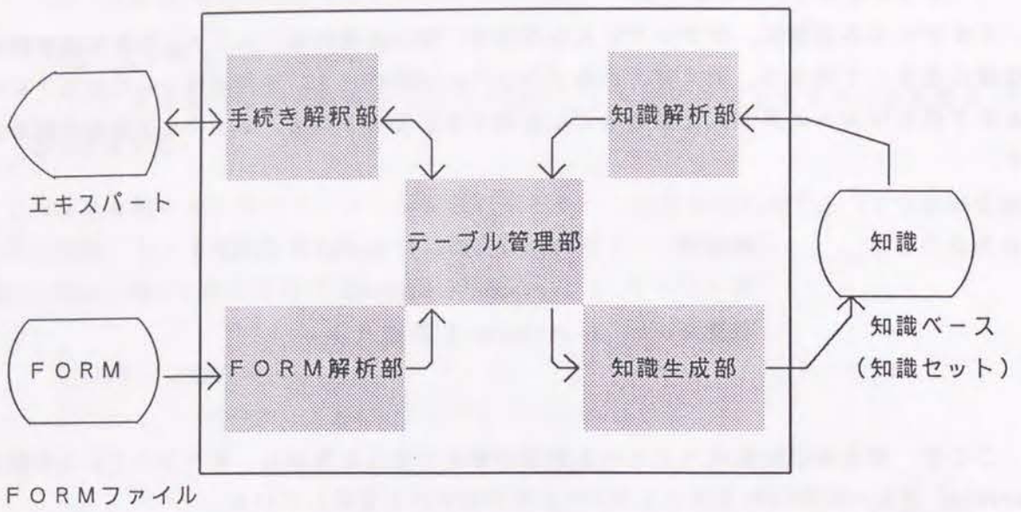


図４．５　ＴＡＩＬＯＲの機能構成

[step3] を実行するための組込手続きとして以下を用意すれば、大半の応用をカバーできると考えている。

（１）put(("<文字列>"|<ブランク>){,}。。。)

ウィンドウに文字列、ブランクに設定されている値を表示する。

（２）get(<ブランク>)

ウィンドウから 文字列を入力し、ブランク値を設定する。

（３）ptn(<子パターン名>,((<子パターン表示構造>){|}。。。)
[,<孫パターン名>,((<孫パターン表示構造>){|}。。。)]。。。)

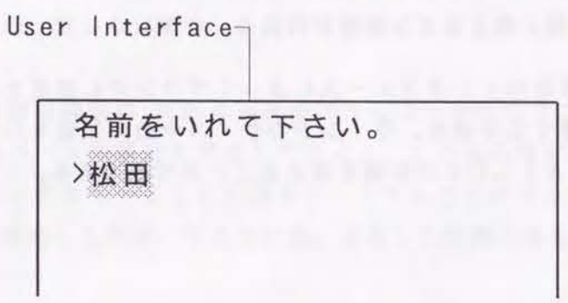
<子パターン表示構造>===
("<文字列>"|<ブランク>|<孫パターン名>){,}。。。

<孫パターン表示構造>===("<文字列>"|<ブランク>){,}。。。

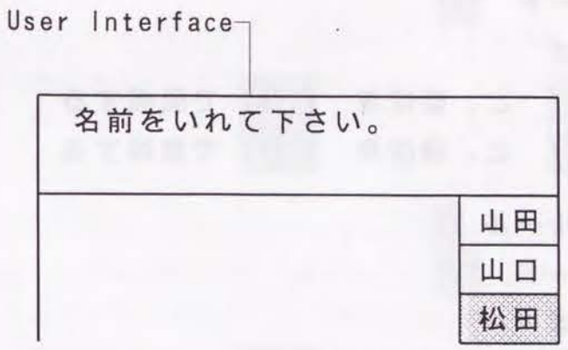
表示構造に従って ブランクに設定された値をウィンドウに表示し、子パターンの保守を誘導する。この表示構造は、エキスパートに自然語風の知識表現を示すためのものである。

put、get 手続きを利用したときの画面表示例を 図４．６に示す。（ａ）では、まず入力の要求メッセージを表示した後、ブランク¯名前に値を入力している。get 手続きは指定されたブランクのファセットに候補が定義されている場合、その候補の一覧をメニューとして表示する。（ｂ）がこの例でブランク¯名前には参加者フレームのインスタンス名のみが入り得ると定義されていると、その候補がポップアップメニューで示されるので入力誤りが起こらない。put、get 手続きの引き数に与えられるブランク名は、親パターン定義の中に指定されたものに限られる。

```
put("名前をいれて下さい。")
get(¯名前)
```



（a）ファセット定義がない場合



（b）ファセット定義がある場合

ﾌｧｾｯﾄ記述 : ¯名前 (instance(参加者))

図４．６　ソーイングメソッドｐｕｔの提示するユーザインタフェース例

子パターン定義、孫パターン定義の中に指定されるブランクは、次の ptn 手続きによりアクセスされる。

```
ptn(‾‾計算ルール, (" 関連度計算ルール", ‾ルール番号,
                    ‾‾‾キーワード, " を含むならば"、‾‾‾関連度加算),
    ‾‾‾キーワード, (" キーワード", ‾キーワード)
    ‾‾‾関連度加算, ("教訓", ‾教訓, "と、確信度", ‾確信度, " で関係する"))
```

この手続きが表示する例を 図４．７に示す。左端に□がある部分から 子パターンに対応する表示、２列目に□がある部分から 孫パターンの表示が始まり、黒白反転部がブランク値を表示する部分である。□ 及び 黒白反転部は、マウスでピックできる部分であり、どこがピックされているかにより 状態遷移がなされ、各状態で 子パターン値、孫パターン値、ブランク値の追加、削除、修正などの操作が行える。

その他にも、テーブル型構造のインタフェースによってブランクに設定された値を表示し、子パターンの保守を誘導する手続き、ウィンドウサイズや色の指定を行なう手続きなどを組み込むことにより、ＴＡＩＬＯＲの価値を高めることが可能である。

User Interface

□関連度計算ルール YY
　□　キーワード　YYY
　を含むならば
　□　教訓　YY　と、確信度　YYYYY で関係する
　□　教訓　YY　と、確信度　YYYYY で関係する

□関連度計算ルール YY
　□　キーワード　YYY
　を含むならば
　□　教訓　YY　と、確信度　YYYYY で関係する

図４．７　ソーイングメソッドｐｔｎの提示する
ユーザインタフェース例

ＦＯＲＭは ある意味で 知識ベースを構造化するので、以下の限定付きの知識検証機能を実現することができる。

（１）重複した知識定義の検出
　繰返し定義の中で、その中の全ての可変部同士の比較により 値が全く同一のものがあれば、それらは重複した知識定義であると警告する。

（２）冗長の知識定義の検出
　繰返し定義の中で、ある一つの可変部同士の比較では一方が他方を含んでおり それ以外の部分が一致している場合、片方が冗長の可能性があると警告する。

（３）矛盾した知識定義の検出
　繰返し定義の中で、ある一部の可変部同士の比較では値が全く同一であり それ以外の部分が一致しない場合、矛盾の可能性があると警告する。

このような検証機能は 知識を逐次追加するような状況では 極めて重要なものである。さらに、知識ベースの構造を同定しておくと、この種の検証機能以外に、繰返し定義の中で、あるブランクをキーとして知識をソートすることができるので、その結果を参照することにより、重複した知識、冗長な知識、矛盾した知識の検出が容易となる。

4．4　計算機システム構成設計支援エキスパートシステムにおける記述例

本節では、これまでに提案したＲＡＮＧＥ、ＦＯＲＭ、ＴＡＩＬＯＲを 計算機システム構成設計支援エキスパートシステム [Tsuji, et al. 1987d] [Iizuka, et al. 1988] に関して記述する例について述べる。本格的にエキスパートシステム構築への適用に関する方式検証は 第５章にて行う。

この構成設計支援エキスパートシステムは 機器構成の間の接続可能性チェック、含めるべきオプションのチェックを行うことを主目的とする。また、選択すべき機器の推薦も行う。構成に ある機器が接続されたり、除去されたとき、あるいは接続方法に変更があったとき、オプション類に変更が必要ないかをルールでチェックする。

システムの専門家モデルを図４．８に示す。このシステムの知識ベースは この図に示すように 役割、知識源の異なる次の４種類の知識から構成される。

（１）機器を表現する知識

これは ハードウェア仕様を表すものである。関連する知識はルール、フレーム、関係で表現される。すなわち、一つ一つの機器は フレームで表現され、機器間の接続可否は ２項関係で表現され、接続に伴うオプションの要否は ルールで記述されている。機器は 逐次旧機種が廃版となり、新機種が登場する。また機種の種類によっては 逐次 機能拡張されうることに注意しておく。

（２）設計図を表現する知識

設計図は 推論を通して作成され 完成される。この知識は フレームで表現されている。設計図の構造は 組織として様式が決められており、大きな技術変革がないかぎり 変更されることはない。設計図の様式が変更になったときは 次の設計手続きも変更する必要がある。

（３）設計の手続きを表現する知識

この知識は 手続きとして記述され、他の知識の利用法に関するものであり、その意味で変更されることは皆無に近い。

（４）ノウハウを表現する知識

この知識は 設計の質を向上するために使われる。例えば、「機器ＹとＺの組み合わせは 拡張性の意味から 機器ＳとＴの組み合わせに変更した方が良い」というような知識である。この種の知識は 熟練したエンジニアだけが所有し、一時に完全なノウハウを獲得することは困難である。すなわち、ステップ・バイ・ステップで蓄積せざるをえない。

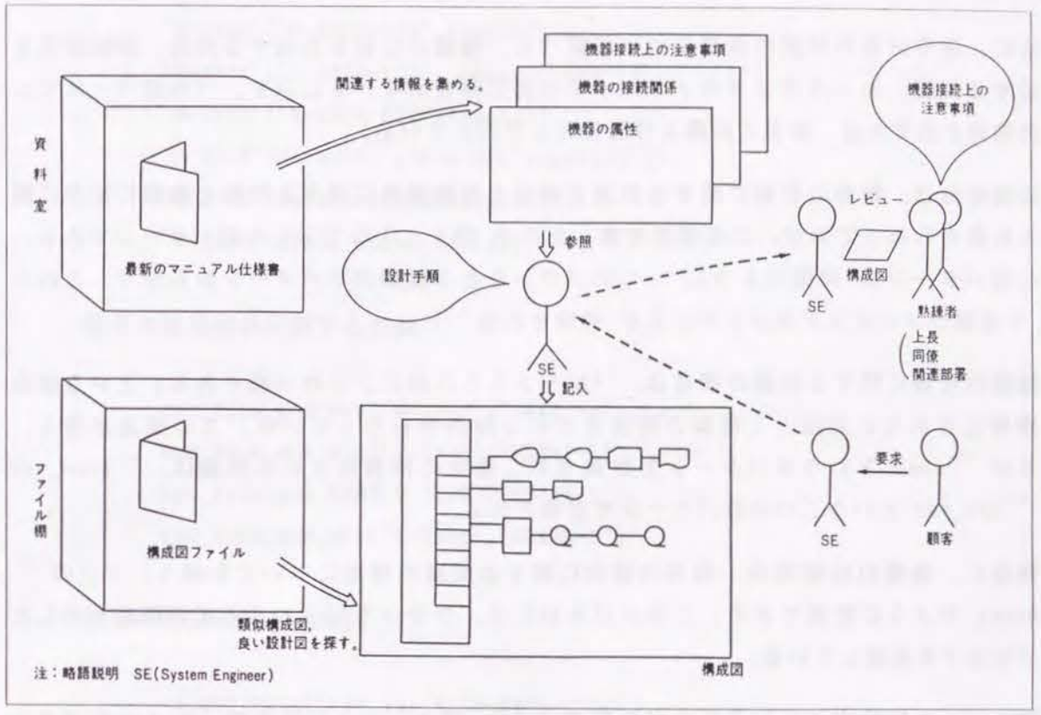


図４．８　計算機システム構成設計支援エキスパートシステムの専門家モデル

Ｒ１[McDermott. 1982] は 我々のシステムと機能的に近いが、Ｒ１が すべてルール形式の知識で記述されているのに対し、我々は 知識のタイプと役割を陽に分類して 種々の知識表現を使いわけるハイブリッド型で知識を記述した。

記述言語は ＰＲＯＬＯＧである。ＰＲＯＬＯＧを用いると 関係、ルール、フレームなど 全て述語により簡便に記述することが出来る [Iizuka, et al. 1988] 。さらに、ＰＲＯＬＯＧのバックトラッキングの機能やユニフィケーションの機能を用いてプログラミングを比較的効率良く進めることが可能である。

しかし、一般の機器の知識を保守する担当者はＰＲＯＬＯＧを直接読んだり理解することは困難である。

新しい機器が登場したり、機能改善されたり、あるいは旧式の機器の販売が中止になることがあるので、機器を表現する知識とノウハウを表現する知識は 継続的に更新される。従って、これらの知識セットに対してのみ ＲＡＮＧＥを定義することにより、エキスパートによる保守対象とする。知識ベースビューのＲＡＮＧＥ定義により、エキスパートは、他の知識セットの存在を意識する必要がなく、一部の知識セットは保護される。

次に、保守対象の知識の構造について調べる。機器の仕様を表現する知識、接続関係を記述する知識、ルールタイプのノウハウの知識の例を図４．９に示す。（知識ベースビューの機能を示すため、本来の知識より簡便化して示している）

知識全体は、機器の仕様に関する知識と機器の接続関係に関する知識と機器の補充に関する知識からなっており、この構造を表したのが 図４．１０で示した親パターンである。この親パターンは 同図のように 一つのブランクと３種類の子パターンからなり、これらは ３種類のソーイングメソッドにより 保守される

機器の仕様に関する知識の構造は、「Ｍ－２２０ＤがＣＰＵの一種である」という構造の階層化されない知識と２種類の階層化される知識からなっている。この構造は図４．１０の --spec という子パターンで定義され、後者の階層化される知識は、--type_var と --opt_var という二つの孫パターンで定義される。

同様に、機器の接続関係、機器の補充に関する知識の構造についても図４．１０の --connect のように定義できる。このＦＯＲＭには、ブランクにとりうる値の範囲を示したファセットも記述している。

図４．１０に示すようにＦＯＲＭを定義すると 図４．１１に示すフィルインザブランクのユーザインタフェースをもつ保守ツールを実現することができる。図から分かるように、エキスパートがアプリケーション向けの表現で 彼に関する知識にアクセスできること、つまり知識ベースビューが提供されていることが確かめられる。

知識の保守には、編集機能だけでなく、知識の検証機能が必要である。これに関し、このシステムは利用時、設計図に含まれる機器を補足したり 置換した場合、システムは 何故そうしたかを説明する。この説明機能により、熟練者は 不正確な知識を抽出したり 知識の不足に発見したりすることが可能である。すなわち、知識ベースを保守する場合、知識を検証するツールは 編集するツールと組み合わせて使用されるべきである。

一方、設計手続きに変更があった場合、そして 設計図のデータ構造がそれに対応して変更される場合、それは 機器やノウハウの知識に責任をもつ熟練ＳＥの仕事でなく システムを設計したナレッジエンジニアの仕事である。つまり、開発環境での知識ベースの再構築となる。

機器の仕様に関する知識

```
is_a('CPU','M-220D').
'M-220D'('H-8820-D22', memory('2MB')).
'M-220D'('H-8820-D22',type(2)).
'M-220D'('H-8820-E22', memory('4MB')).
'M-220D'('H-8820-E22',type(3)).
'M-220D'(default, console['H-8802-2']).
'M-220D'(default,segment_no(6001)).
'M-220D'(capa, no_of_mtu(2)).
```

機器の接続関係に関する知識

```
cpu_type_and_mtu('M-220D','H8467-A').
cpu_type_and_mtu('M-220D','H8467-1').
cpu_type_and_mtu('M-220D','H8468-1').
cpu_type_and_mtu('M-220D','H8426-1').
```

機器の補充に関する知識

機器の補充に関する知識

```
ce_mtu_rule((ret_var(X,'H-8426-1',Z)
             number_of_units(X,0),
             Z>0
            ==>
            set_ce_mtu(X,'H-F8426-11']),
            set_option(X,['H-F8426-50']))).

ce_mtu_rule((ret_var(X,'H-8426-1',Z)
             number_of_units(X,1),
             Z>1
            ==>
            set_ce_mtu(X,'H-F8426-12'))).
```

図４．９　計算機システム構成設計支援エキスパートシステムの知識表現の例

```
form 機器を表現する知識の型
  pattern{
     /* ~作成年月日 */
     ~~spec            /* 機器の仕様     */
     ~~connect         /* 機器の接続関係 */
     ~~option          /* 機器の補充     */
       }
  cpattern{
    ~~spec    (
               is_a('CPU',~cpu).
               ~~~type_var
               ~~~opt_var
              )。。。
    ~~connect  (cpu_type_and_mtu(~cpu,~mtu).)。。。
    ~~option   (
               ce_mtu_rule((ret_var(X,~equip,Z),
                            number_of_units(X,~no_of_equip),
                            Z>~no_of_equip
                           ==>
                            set_ce_mtu(X,~equip_name).
                            ~~~opt_rule)).
              )。。。
         }
  gpattern{
    ~~~type_var       (~cpu(~type,memory(~m_spec MB)).
                       ~cpu(~type,type(~t_spec))
                      )。。。
    ~~~opt_var        ([(~cpu(default,console(~con_name)). ]
                      |[ ~cpu(default,segment_no(~seg_no)).]
                      |[ ~cpu(capa, no_of_mtu(~no)).       ]
                      )。。。
    ~~~opt_rule       (  [set_option(X,[~opt_name])]  )。。。
}
```



```
facet   {
  ~no_equip      (type(int), range[0,10])
  ~no            (type(int), range[0,10])
  ~opt_name      (instance(option_equipment))
  ~equip_name    (instance(equipment))
  }
sewing-methods{
  spec_mainte(){
   put("本日の日付をいれて下さい。")、
   get(~作成年月日)、
   ptn(~~spec, ("ＣＰＵ名", ~cpu, "には以下の種類があります："
                          ~~~type_var
              "さらに次の仕様です：", ~~~opt_var),
      ~~~typevar, (~type, "は、標準ﾒﾓﾘ", ~m_spec, "MB, タイプ", ~t_spec)
      ~~~optvar,  ( ("コンソールの標準   ：", ~con_name)
                  |("セグメント数の標準：", ~seg_no)
                  |("接続可能ＭＴＵ台数：", ~no)      )
    )
   }
  connect_mainte(){
   put("本日の日付をいれて下さい。")、
   get(~作成年月日)、
   ptn(~~connect, ("ＣＰＵ", ~cpu, "は、ＭＴＵ", ~mtu, "と接続可能")
      )
   }
  option_mainte(){
   put("本日の日付をいれて下さい。")、
   get(~作成年月日)、
   ptn(~~option, ("構成に", ~equip, "を", ~no_of_equip, "個含むならば",
                  "今回追加の名称を", ~equip_name, "とする", ~~~opt_rule),
      ~~~opt_rule, ("オプション" opt_name, "を加える"  )
      )
   }
}
```

図４．１０　計算機システム構成設計支援エキスパートシステムのＦＯＲＭ記述例

機器の仕様に関する知識

□CPU名　M-220D　には以下の種類があります：
　□　H-8820-D22　は標準ﾒﾓﾘ　2　MB　タイプ　2
　□　H-8820-E22　は標準ﾒﾓﾘ　4　MB　タイプ　3
　さらに次の仕様です：
　□　コンソールの標準　：　H-8802-2
　□　セグメント数の標準：　6001
　□　接続可能ＭＴＵ台数：　2

機器の接続関係に関する知識

□ＣＰＵ　M-220D　は、ＭＴＵ　H8467-A　と接続可能
□ＣＰＵ　M-220D　は、ＭＴＵ　H8467-1　と接続可能
□ＣＰＵ　M-220D　は、ＭＴＵ　H8468-1　と接続可能
□ＣＰＵ　M-220D　は、ＭＴＵ　H8426-1　と接続可能

機器の補充に関する知識

□　構成に　H-8426-1　を　0　個含むならば
　　今回追加の名称を　H-8226-11　とする
　□オプション　H-8426-50　を加える
□　構成に　H-8426-1　を　1　個含むならば
　　今回追加の名称を　H-8226-12　とする

図４．１１　計算機システム構成設計支援エキスパートシステムの知識保守のユーザインタフェース例

（注）左図において
　□　を単位として、挿入、削除などの編集が可能
　□　の組にたいし、値の包含関係の検出が可能
　　を単位として、値の修正、変更が可能
　　ごとにソートが可能

４．５　まとめ

本章では、第３章の構想に基づき、知識ベースを静的に保守するための知識ベースビューついて論じた。

一度エキスパートシステムの目的が決まり 要求が明確になると、それまでに記述した知識には構造があるではないか と仮説をたてた。本章の方式の原点は、この構造をナレッジエンジニアに陽に認識させ、それをＦＯＲＭとして定義させることにより、ナレッジエンジニア用の知識表現とエキスパート用の知識表現を相互に変換することにある。この知識表現の抽象レベルの変換により、エキスパート自らによる知識保守を可能にした。

本方式に基づき生成される知識ベースエディタＴＡＩＬＯＲは、ＦＯＲＭを解釈することにより、知識ベースビューとして、エキスパートが知識ベースのどの部分にアクセスできるかを示し、基本的に変更してはならない部分を隠蔽する。アクセスできる部分に対しては、フィルインザブランク形式でエキスパートの操作を誘導し、重複した知識や冗長な知識の警告を行う。

ＦＯＲＭを定義するだけで、ナレッジエンジニアが構築した知識ベースをエキスパートが保守できるようになる。従って、本章で述べた知識ベースビューは、新たな問題解決モデルが考案される都度生じていた個別に知識獲得ツールを開発する労力を省力化した。

本論では、ＦＯＲＭの表現手段の設計に当り、知識ベースの構造を３階層に限定した。応用によっては４階層以上の複雑な定義手段が必要となる場合もあるかもしれない。階層に関して方式の拡張性に課題はないが、４階層以上となった場合 エキスパートによる知識の保守はＴＡＩＬＯＲを用いても困難になると考えている。

知識ベースにとって ＦＯＲＭは必須ではない。エキスパートに自ら知識ベースを保守させたいという要求が出た時点で 後から定義すれば良い。この意味で ＦＯＲＭ定義が そもそも知識ベースの持つ柔軟性を損なうことはない。さらにＦＯＲＭは、記述例からも分かるように 保守対象の知識表現をもとに定義することができる。従ってＦＯＲＭの定義はナレッジエンジニアにとって負担とはならない。

記述例で示した構成設計問題は、エキスパートシステムの一つの類型を示すものである。このような類型化が複数でき、標準的なＦＯＲＭを見出すことができれば、エキスパートが自ら新しく知識ベースを作る試行環境も実現できる。

第５章　知識ベースの静的保守方式の適用と検証

５．１　まえがき

本章では、第４章で提案した知識ベースビューをプログラミングノウハウを伝承するエキスパートシステムに適用し、試行、開発、利用、保守環境がいかに構築されるかを論じる。

このエキスパートシステムは １６１件のソフトウエア事故に関する教訓的なノウハウ（１件１葉形式の文書）を蓄積しており、本エキスパートシステムの問題は 新たに生じた事故の背景にある原因を既知の教訓に結び付けることである。具体的な事故と格言化した教訓では それらの抽象度が異なるので、単純なキーワード検索で 両者を結合することは困難である。一方、教訓を特徴付ける属性の設定が困難であるため、分類問題に帰着することも困難であった。

そのため、問題解決手法（専門家モデル）として 具体的な失敗から得られるキーワードを抽象化する連想ルールと、抽象化したキーワードと教訓を結び付ける関連ルールとを記憶しておき、初心者が いくつかの具体的なキーワードを入力すると 関連するプログラミングノウハウを検索して表示する ものを採用した。本章では、この専門家モデルのことを連想検索と呼ぶ。

知識ベースの構築は 次の３段階を踏んだ。

（ｉ）エキスパートから思い付くまま連想ルールと関連ルールを得る。
（ii）初期化した知識ベースによる検索結果とエキスパートが判定した結果とを
　　比較することにより ルールの洗練化を行う。
（iii）ルールの保守ツールをエキスパートに提供する。

完全なルール群は短期間では得られない。そこで、エキスパートが気付いたときに新たなルールを追加できることが重要と考え、知識ベースビューを（iii）の段階で設けた。構築したエキスパートシステムがもつ専門家モデル（連想検索モデル）は 本エキスパートシステムだけでなく他の応用分野にも使える。これに着目して、本章では、設計した知識ベースビューは 保守のみでなく 別のエキスパートシステムの試行に利用できることも示す。

５．２　検索型エキスパートシステムＳＯＣＫＳの推論モデル

５．２．１　ＳＯＣＫＳを開発する背景

近年、コンピュータ・システムの開発において、ソフトウェア開発の比重が非常に増大している。それとともにソフトウェアの品質を向上するために種々の手法が研究されている［Nara, et al. 1984］。

このような背景のなかで、品質向上の一つの施策として プログラミングノウハウの伝承が考えられる。プログラミング作業の中には、ループの判定の仕方、共通エリアの使用法など多くの常識ともいうべきものがある。これらに熟知していないために、あるいは遵守しなかったために 設計段階でバグが埋め込まれ、検査段階でそれが摘出されず、運用段階において大きな障害として現れることも少なくない。逆にいうと、先の常識を遵守しさえしていれば、障害の原因は未然に除去できる場合が多い。従って、ソフトウェア開発における常識を設計者に熟知させることが重要である。

著者の周辺のソフトウェア開発部署では、このようなプログラミングノウハウを教訓集としてまとめている。これは ソフトウェア常識集（以下、単に常識集と略す）と呼ばれ、１３分類のもとに１６１件の教訓からなっている。教訓と事例は１件１葉に纏められている。この例を 図５．１に示す。

「ソフトウェア障害の８割以上が １６１件の教訓のいずれかに該当する」と言われており、多くの設計者は、この常識集を手許において、業務を遂行している。常識集は、現在個人の自己啓発、小集団活動などグループ勉強会、講習会などの教育、ＯＪＴに使われている。

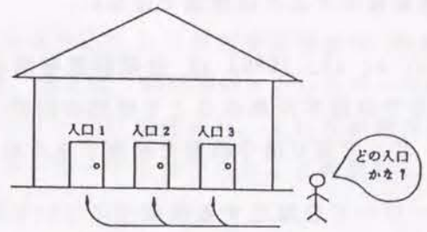
今回、常識集にまとめられている教訓を、さらに周知徹底するためにエキスパートシステム化した。このシステムは、ＳＯＣＫＳ(Software Common Knowledge Transfer System)と呼ばれ［Tsuji, et al. 1987c］、エキスパートシステム構築ツールＥＳ／ＫＥＲＮＥＬ［Yoshimura, et al. 1988］により実現されている。本章では、このＳＯＣＫＳの設計思想と４章で提案した知識ベースビューの適用について述べ、その有効性と限界について言及する。

５．２．２　従来ＩＲシステムとの違い

常識集の教訓は、個々のソフトウェア設計例や障害例（以下まとめて事例と呼ぶ）から見ると 抽象度が高い。従って、未熟練者が 短時間で この１６１件の教訓に習熟することは容易ではない。しかし、経験的にある特定の事例に対し 教訓を結び付けると、印象深く その教訓を学習できることが知られている。

目次の一部

分類	No.	題　　名	備考	ページ
	1	一つのルーチンは入口を一つにせよ	初期設定	1
	2	同一処理は１か所で行え		2
	3	エラーチェック／リトライは２回以上行え	エラー	3
	4	Ａでなく，ＢでないならＣとは限らない		4
			るな	5
			境界／限界	6
			エラー	7
			をチェックせよ	

題名	一つのルーチンは入口を一つにせよ	1. 流れの処理（初期設定）
教訓・要点	処理を簡略にするため，一つのルーチンにいくつかの入口を作ることがある。このような場合，ほとんどが類似した入口名を付けるため，入口を間違えやすい。玄関を一つにし，行きたい部屋に行きやすいようにしておく方が来客に対して親切である。	

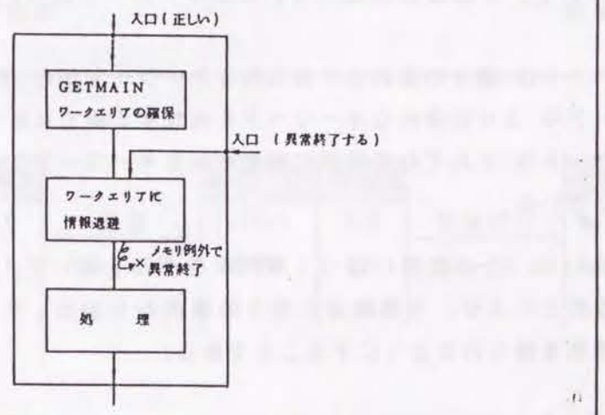
事例	正しくない入口からはいって異常終了	品名	オペレーティング・システム
1 現象	ルーチンの入口が二つあって，先頭からはいった場合は正しく処理されるが，途中からはいった場合は，GETMAIN によるワークエリアの確保が抜けてしまい，エリアを破壊しメモリ例外で異常終了となった。		
2 原因			
3 対策	正しい入口から入れるよう入口を一つにした。		

図５．１　ソフトウェア常識集の教訓の例

各教訓の内容は，個々の事例を抽象化した絵入り文書にとり纏められている

ところが、ある一つの事例にたいし、いかなる教訓が該当するかを見極めようとするとき、図５．１に示した目次を見ても、利用者は「どの教訓が自分の事例に該当するのか」を判定しにくいという問題がある。これまでは、設計担当部署のリーダが個別の事例ごとに いかなる教訓に関連するかを 未熟練者に指導していた。

ある障害事例を１６１件のどの教訓に結び付けるかは 一種の分類問題である［Clancey. 1984］。ＥＴＳ［Boose. 1985］は 分類対象を３つづつ選択してきて それらを区別する属性、その違う度合いを インタビューにより 獲得する知識獲得ツールである。しかし、教訓にそのような属性を見出すことは容易ではない。

ＣＳＲＬ［Bylander, et al. 1986］は 分類問題専用の知識表現言語を提供する。ここでは、分類していく上で分類木があることを暗黙の前提としており、医療診断分野に適用されてきた。しかし、ここで取り扱う教訓を整理するための分類木はなかった。

一方、各教訓にキーワードを設定する検索法について考えてみる。この場合、教訓のキーワードと個々の事例のキーワードとが 正確に一致しなければならない。例えば、事例のキーワードとして「ＥＯＦ(End of File)」を抽出したとき、「境界」あるいは「終了判定」など より抽象的なキーワードが与えられている教訓は 検索できない。さらに、従来のキーワード検索では、個々のキーワードは 検索対象に対して 設定できるか否かの２値でしか設定できなかった。しかし、キーワードによっては ある教訓に密接に関連する一方、別の教訓には少し関係する という場合も少なくない。

以上の分析に基づいて、常識集に精通しているエキスパートについて、次の仮説を立てた。

（１）エキスパートは 個々の事例より得られるキーワードから それに関連するキーワードや より抽象的なキーワードを連想する能力を有す。

（２）エキスパートは それぞれの教訓に設定されるキーワードとそのウェイトを既知としている。

ＳＯＣＫＳのねらいは、この仮説に基づく専門家モデルを用いて、エキスパートの知識を収集し 記憶することにより、未熟練者が個々の事例から抽出したキーワードを入力するだけで該当する教訓を得られるようにすることである。

エキスパートが行なう常識集の検索を整理すると 次のようになる。

（１）事例からキーワード（複数）を抽出する。

（２）抽出したキーワードから、重み（これをキーワードの確実度と呼ぶ）を付けて同じ意味のキーワード、関連するキーワード、より抽象的なキーワードを連想する。

（３）キーワードから 重み（これを関連度と呼ぶ）を付けて教訓を関連付ける。

（４）関連度の高い教訓を検索結果とする。

キーワードを抽出するとき、不十分であったり 不正確であることもありうるので関連度の最も高いもの あるいは しきい値以上のものが、必ずしも障害に対応する教訓となりえないことに注意する。

この検索方式をエキスパートシステムとして実現する場合の 利用者とシステムの役割分担を図５．２に示す。この図は、例えば、利用者が入力したキーワード「装置」「待ち」から「デッドロック」というキーワードが連想され、それが教訓Ｎｏ．８，１２，２３などに関連付けられ、入力されたキーワード全体からみると教訓Ｎｏ．２３が最も関連が強いであろうことを示している。

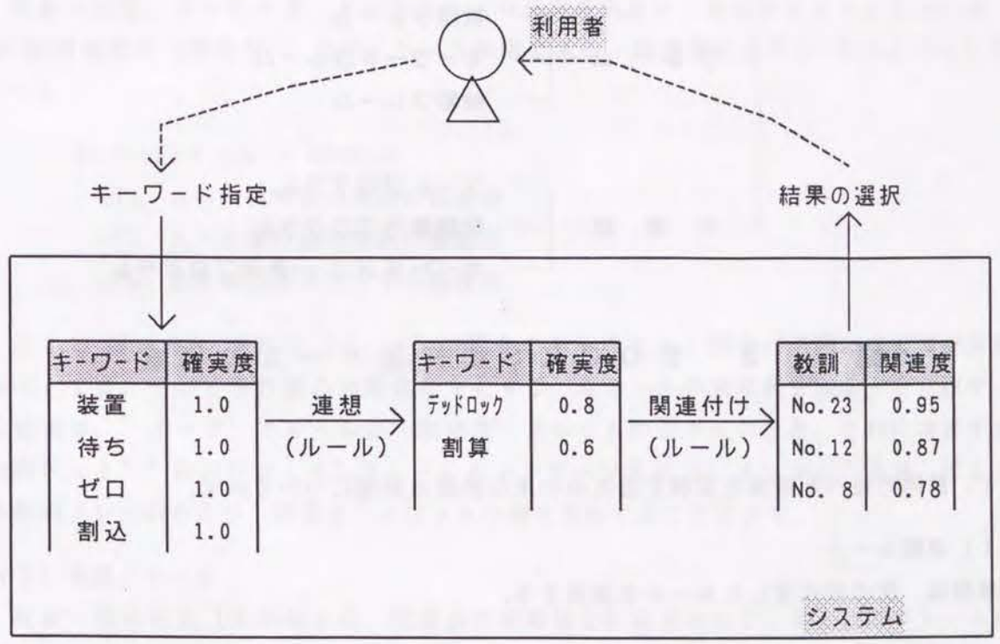


図５．２　問題解決の専門家モデル

５．３　ＳＯＣＫＳの知識ベースの構成

５．３．１　知識表現

知識ベースのインプリメントに利用したエキスパートシステム構築ツールＥＳ／ＫＥＲＮＥＬは、知識モデルとしてフレーム、ルール、手続き（Ｃ言語）、およびメタルールを提供する。これらを用いて実現したＳＯＣＫＳの知識ベースの構成を図５．３に示す。（ＥＳ／ＫＥＲＮＥＬの知識表現及び推論アルゴリズムの概要については付録に示す）

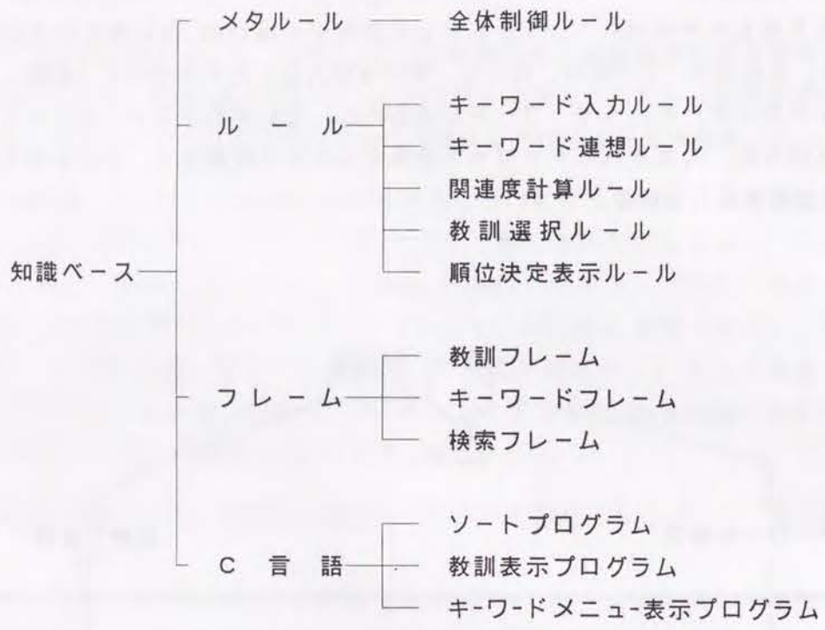


図５．３　ＳＯＣＫＳの知識ベースの構成

以下、前節で述べた検索を実現するための主な知識の表現について示す。

（１）連想ルール

連想は、次の形式をしたルールを使用する。

```
if  キーワード1, キーワード2, ．．．
        then  キーワードn （確実度m）
```

本ルールは、キーワード$_1$，キーワード$_2$が利用者により指定されたならば、キーワードnの確実度を m 上げるという意味である。確実度の計算は 次のようにして行なった。


```
cfn = cfo + cfm － cfo*cfm
    cfn  ルール実行後のキーワードmの確実度
    cfo  ルール実行前のキーワードmの確実度
    cfm  条件部のキーワードの確実度の最小値と
            ルールに付加されている確実度mの積
```

ここで 確実度は 便宜上［０，１］の値をとるものとし、度合いが強いほど値が大きいものとする。そして 利用者が選択したキーワードの確実度を１とし、選択しなかったものの初期値を０とする。ＥＳ／ＫＥＲＮＥＬで連想ルールを表現した例を図５．４に示す。この表現は”連想検索”フレームの”入力用語”スロットに”無限ループ”を含むとき”ループ”フレームと”判定もれ”フレームにメッセージを送り、メソッド”重み加算”によりキーワード”ループ”および”判定もれ”の確実度を更新することを示す。

（２）関連度計算ルール

関連付けのルールの形式は 次である。

```
if  キーワード1
        then  教訓1  （関連度1），
              教訓2  （関連度2）。。。
```

本ルールは、キーワード$_1$ が利用者により指定されるか、連想されるかしたならば、教訓$_1$の関連度を（関連度$_1$）上げるという意味である。関連度の計算は 次のようにして行なった。

```
cfn = cfo + cfm － cfo*cfm
    cfn  ルール実行後の教訓の関連度
    cfo  ルール実行前の教訓の関連度
    cfm  条件部のキーワードの確実度
```

ここで 関連度も 便宜上［０，１］の値をとるものとし、度合いが強いほど値が大きいものとする。そして各教訓の初期値を０とする。本ルールの表現例を図５．５に示す。この表現は、”ループ”フレームの”関連度”スロットの値が正のとき、それに関連する”教訓Ｎｏ１””教訓Ｎｏ１４”フレームにメッセージを送り、メソッド”加算”により”各教訓フレームのもつ”関連度”スロットの値を更新することを示す。

（３）教訓フレーム

教訓一般の性質（表示の方法、関連度の初期値と計算方法など）を上位のフレームに記述し、個々の教訓の属性（タイトル、キーワードなど）を下位のフレームに記述する。この記述例を図５．６に示す。本表現は、各教訓フレームは”教訓”の呼ぶフレームの下位フレーム（インスタンス）であり、タイトル、分類１、分類２、キーワード、文書ファイルｉｄなどのスロットをもつことを示している。

```
(キーワード連想ルール 10
if
  (連想検索の @入力用語 が 無限ループ を要素とする)
then
  (send ループ    重み加算(0.8))
  (send 判定もれ 重み加算(0.6))
)
```

この表現は”連想検索”フレームの”入力用語”スロットに”無限ループ”を含むとき”ループ”フレームと”判定もれ”フレームにメッセージを送り、メソッド”重み加算”によりキーワード”ループ”および”判定もれ”の確実度を更新することを示す。

```
(キーワード連想ルール 20
if
  (連想検索の @入力用語 が ファイル を要素とする)
then
  (send データ消失 重み加算(0.7))
)
```

図５．４　連想ルールの例

```
(関連度計算ルール 10
if
  (ループ の @関連度 を ?x  とし
           @関連度 が   0  より大きい)
then
  (send Ｎｏ1  加算(0.7,?x))
  (send Ｎｏ14 加算(0.6,?x))
)
```

この表現は、”ループ”フレームの”関連度”スロットの値が正のとき、それに関連する”教訓Ｎｐ１”“教訓Ｎｏ１４”フレームにメッセージを送り、メソッド”加算”により”各教訓フレームのもつ”関連度”スロットの値を更新することを示す。

```
(関連度計算ルール 20
if
  (判定もれ の @関連度 を ?x  とし
           @関連度 が   0  より大きい)
then
  (send Ｎｏ1  加算(0.8,?x))
  (send Ｎｏ12 加算(0.4,?x))
)
```

図５．５　関連度計算ルールの例

（４）キーワードフレーム

キーワード一般の性質（確実度の初期値と計算方法など）を上位のフレームに記述し、個々のキーワードの属性を下位のフレームに記述する。この記述例を図５．７に示す。本表現は、”関連度”スロットは実数タイプの値をとり、初期値が０であることを示している。

その他の キーワードの入力処理や教訓の表示処理など入出力処理は、Ｃ言語により、手続き的に記述した。

```
(教訓                                    /*上位フレーム*/
 super_class     system
 タイトル        (data_type    string)
    . .                      . .
    . .                      . .
)
(Ｎｏ２５                                /*下位フレーム*/
 class           教訓
 タイトル        ループ処理判定位置には、充分注意せよ
 分類１          流れの処理
 分類２          境界・限界
 キーワード      {  Ｋ１，Ｋ２，．．．  }
 文書ファイルｉｄ  ２５
)
```

ここで”Ｎｏ２５”はフレーム名、”タイトル”、”分類１”などはスロット名、スロット名に続く文字列や値はスロット値を表す。

```
(Ｎｏ２６
 class           教訓
   . .             . .
```

図５．６　教訓フレームの例

```
(キーワード                              /*上位フレーム*/
 super_class     system
 関連度          (data_type    real)
                 (initial      0.0 )
)
(ループ                                  /*下位フレーム*/
 class           キーワード
)
(判定もれ
 class           キーワード
)
```

ここで、”ループ”、”判定もれ”などはフレーム名であり、各下位フレームはキーワードフレームの関連度スロットをもつことを示す。

図５．７　キーワードフレームの例

5．3．2　SOCKSの入出力

SOCKSは、プログラミングの対象、解法や障害の現象、原因などに関するキーワードを入力とし、それらのキーワードと関連する教訓を順序付けて出力する。操作の簡便化を図るためキーボード操作を削減し、大半をマウス操作で検索できるようにした。

キーワードを入力する画面例を 図5．8に示す。利用者は、メニューの中から任意の数のキーワードをマウスでピックする。キーワード数が 約４００と多いため、上下／ジャンプスクロールだけでなく、５０音指定によるスクロール、分類区分にそったキーワードメニュー表示を行ない、利用者インタフェースの向上を図っている。

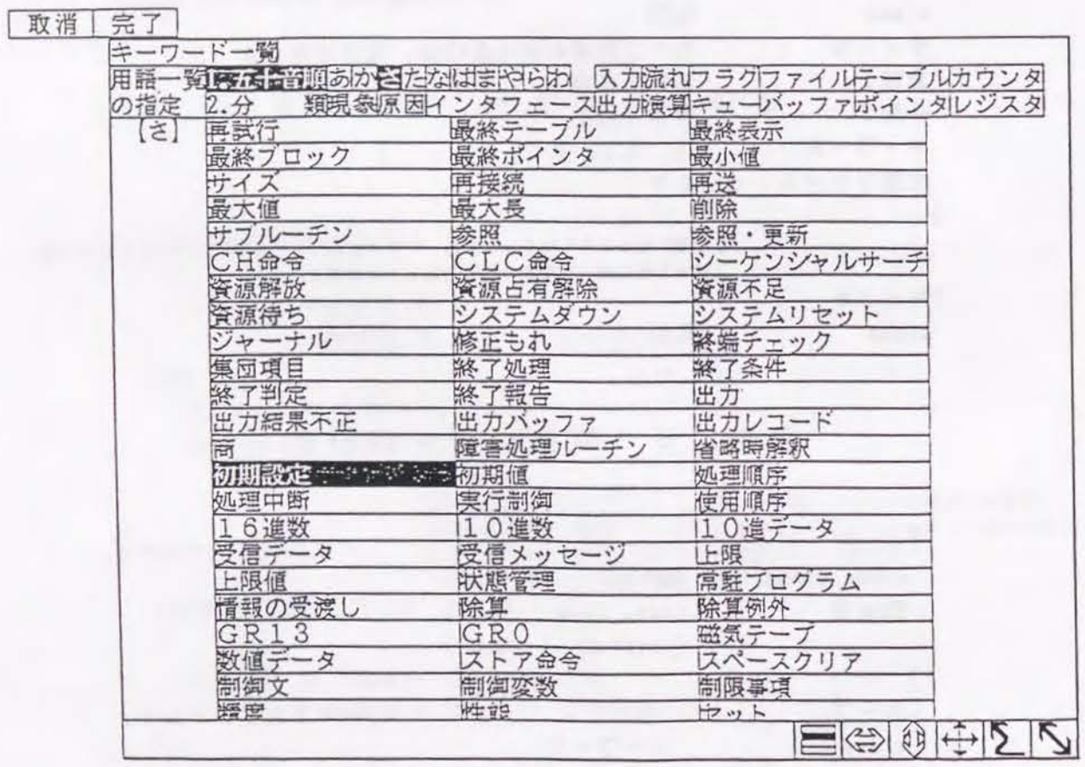


図５．８　ＳＯＣＫＳのキーワード指定画面例

上下／ジャンプスクロール、５０音指定によるスクロール、
分類区分にそったキーワードメニュー表示を行う。

検索結果は、関連度に関し降順に表示する。利用者は、マウスの操作により、関連度に関し前後関係にあるものを順次表示することができる。この表示例を図５．９に示す。図のように 教訓は文書処理ソフトウェアにより絵混在文書として記憶されている。

ＳＯＣＫＳは、関連度がある値以下の教訓、検索順位がある値以下の教訓を 検索結果としない打切り機能を有しており、結果が大量になることを抑制している。さらに選択したキーワードから連想したキーワードを確認できるだけでなく、その連想を抑制することもできる。

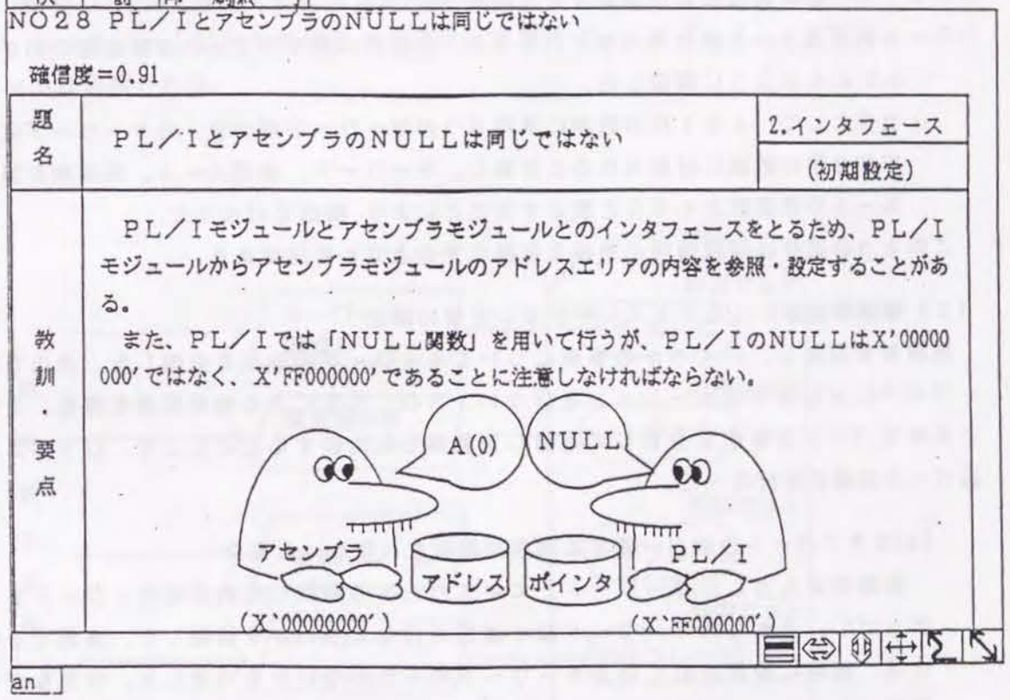


図５．９　検索結果の表示画面例

関連度の高いものから順次、絵入り文書で表示する。

５．４　知識ベースビューのＳＯＣＫＳへの適用

５．３で示した連想ルール、関連度計算ルール、教訓に設定するフレームなどは、ソフトウェア検査のエキスパートが分担して作成した。分担により 作業日数を短縮できる反面、整合性に問題（重複、矛盾など）が生じることがある。また、ルールの抽出に漏れがある可能性も高い。さらに 連想ルールに付与される確実度および関連度計算ルールに付与される関連度は主観的な値であるため、後日変更が必要となる場合もある。

従って 教訓数、ルール数が増大するとともに全体的に知識ベースを調整する必要がある。ＳＯＣＫＳの知識ベースの開発においては、以下の手順で知識の洗練化を行なった。

（１）初期設定

ルールを体系だてて設定することは困難であるため、ルールに不足があったり、確実度、関連度に多少の誤りがあることも認めて抽出作業を進めることが良いと考えられる。ここでは、知識ベースの一貫性の保持にはあまりとらわれることなく、次の基準を設定して知識獲得を行った。

- すべての確実度および関連度を５段階（0.1，0.3，0.5，0.7 ，0.9）に分類した。
- 各教訓は３～８個のキーワードを与え、各教訓に割り当てられる関連度の和が 3.5 になるように調整した。
- 目安として、１６１件の教訓に平均５つのキーワードがつき、各キーワードは平均２件の教訓に付加されると計算し、キーワード、連想ルール、関連度計算ルールの目標数を４００と設定することにより 抽出を行なった。

このような基準は知識獲得の進捗を可視化する上でも有効である。

（２）被験者設定によるシミュレーションと値の補正

被験者を設定し、いくつかの事例についてキーワードの抽出を依頼した。抽出されたキーワードに対しシミュレーションを行ない、ＳＯＣＫＳによる検索結果を得た。この結果とエキスパートが事例を分析して指摘した教訓とを比較することにより、以下のように知識ベースの補正を行なった。

(a)エキスパートの解と一致する教訓が検索されなかった場合

被験者が入力したキーワードとエキスパートの選択した教訓のキーワードをリストアップし、それらのキーワード間で連想が行なえないかを検証した。連想できない場合は、教訓に設定されているキーワードの不足がないかを吟味した。当然ながら、被験者のキーワード設定が不適切な場合もあった。

(b)エキスパートの解と一致する教訓が検索されたが、順位が低かった場合

被験者が入力したキーワードと エキスパートの選択した教訓のキーワードおよび上位に検索された教訓のキーワードをリストアップし、その間の確実度、関連度の値の適否を調べた。被験者の選択したキーワードが適切であった場合を取り上げ、エキスパートの選択した教訓が上位に検索されるように 値を変更した。このフェーズでは、値を５段階に分類するという制約を取り除いた。

（３）保守環境の設定

（２）の補正を行なってもルールの抽出が不充分であったり、確実度、関連度が不適切に付加されているルールが残存している可能性がある。ＳＯＣＫＳの開発では、不十分、不適切な知識は、長期にわたり部分的に残らざるをえないと考え、（２）の段階をある時期に打ち切った。知識ベースを短期間で完全にすることを放棄し、そのかわりに第４章で述べた知識ベースビューにより エキスパートが継続的に知識ベースを保守できる環境を設定した。

以下、設定した環境について説明する。ここでの目標は、ルールやフレームなどの知識モデルでなく、問題解決を表す図５．２に対応する専門家モデルを使用して 知識ベースの保守を可能とすることである。

まず、知識ベースのうち、継続的に保守される知識を限定する必要がある。ＳＯＣＫＳの例では 図５．３に示した知識セットの中で、キーワード連想ルール、関連度計算ルール、教訓フレーム、キーワードフレームの４種のセットが、専門家モデルに陽に表れるものである。図５．１０に示すようにこれらの知識セットにたいしてのみＲＡＮＧＥ定義をすると、エキスパートは他の知識セットの存在を意識する必要がなく、そのため、一部の知識セットは保護される。

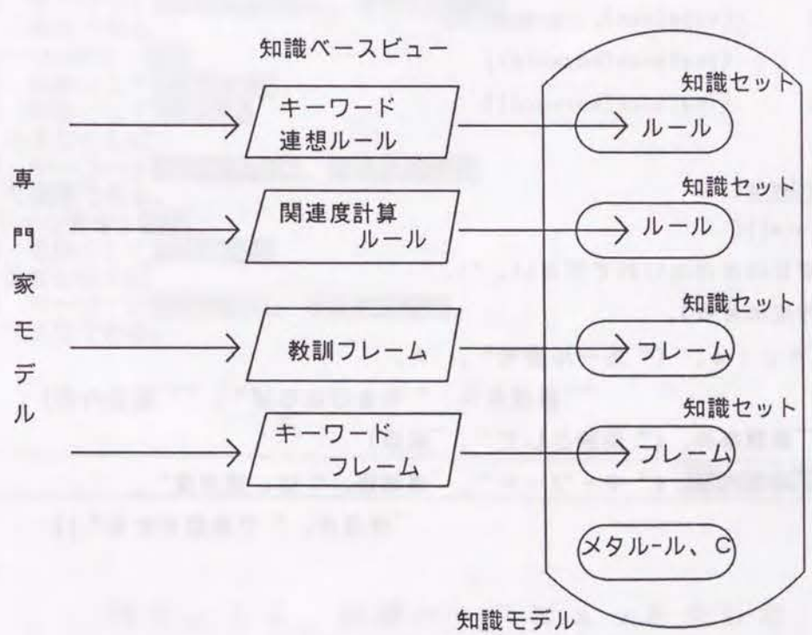


図５．１０　知識ベースビューによるＳＯＣＫＳの保守

ビュー定義されない知識セットは保護される

```
form 連想ルールの型
  pattern{
     /* ~作成年月日 */
     (キーワード連想ルール)
     ~~Ｒｕｌｅ
       }
  cpattern{
    ~~Ｒｕｌｅ   (
                  (キーワード連想ルール~A)
                  if
                    ( 連想検索 の ~~~連想条件
                  then
                    ~~~連想内容
                  ) 。。。
        }
  gpattern{
    ~~~連想条件     (  @入力用語  が  ~用語  を要素とする) 。。。
    ~~~連想内容     (send  ~連想語  重み加算(~確信度) )
    }
  facet   {
    ~確信度        (type(num), range[0,1])
    ~用語          (instance(keyword))
    ~連想語        (instance(keyword))
    }
  sewing-methods{
    rulemainte(){
     put("本日の日付をいれて下さい。")、
     get(~作成年月日)、
     ptn(~~Ｒｕｌｅ, (" ルール番号", ~A,
                          ~~~連想条件, " を含むならば", ~~~連想内容),
         ~~~連想条件, (" 用語として", ~用語)
         ~~~連想内容, (" キーワード", ~連想語, " は、確信度",
                                        ~確信度, " で連想できる" ))
     }
}
```

図５．１１　連想ルールにたいするＦＯＲＭ定義の例

次に、保守対象となる４つの知識セットにエキスパートがアクセスするためには、いかなる部分がいかなる値で修正できるかを可視化して提示する必要がある。

キーワード連想ルールの表現例である図５．４に注目すると、

・不特定数のルールから知識セットが形成されていること、

・各ルールの条件部、行動部には構造が存在すること、

・キーワードや重みなど設定される値に制約があること、

を確認することができる。

従って、これらの３つの要件を、それぞれ図５．１１に示す子パターン、孫パターンとファセットでＦＯＲＭを定義することができる。さらに、同図のようにソーイングメソッドを定義すると、ＴＡＩＬＯＲは 図５．１２に示す自然語風のインタフェースを エキスパートに示す。ここで、□を単位として、追加・削除ができ、黒白反転部に値の代入がなされる。

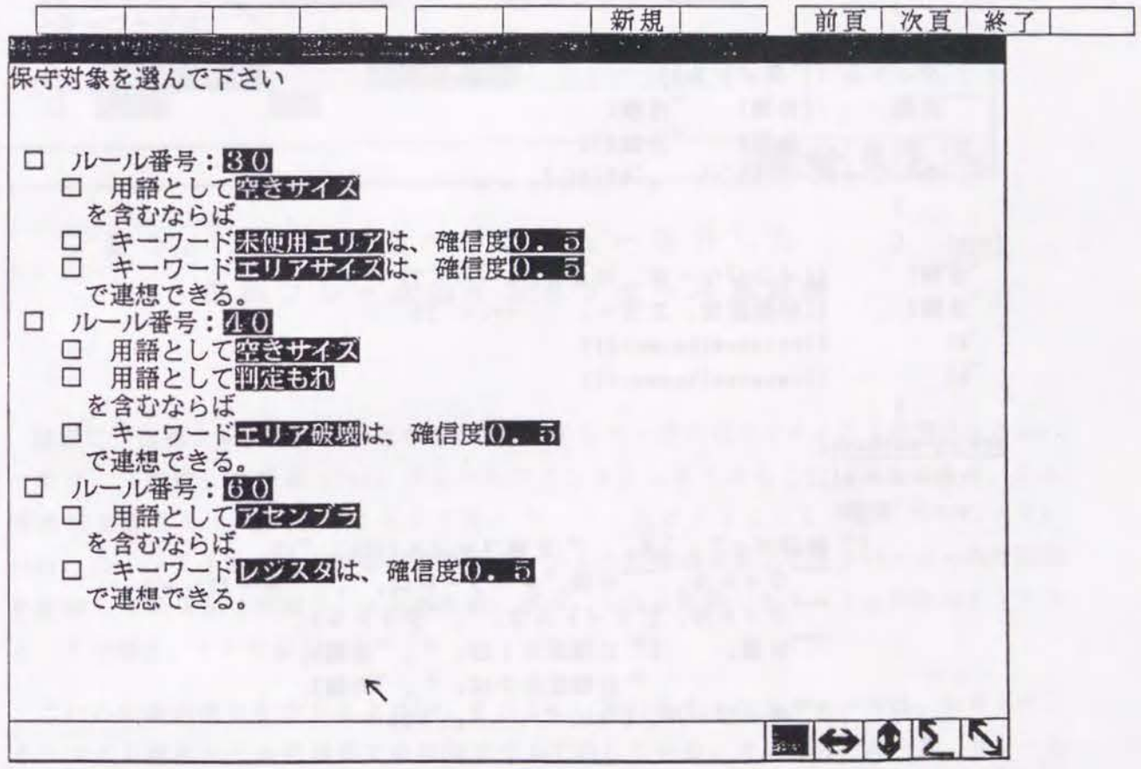


図５．１２　知識ベースビューを介した
連想ルールのインタフェース画面例
黒白反転部が値を代入できるブランクである

```
form 教訓
  pattern{
      ～［固定部］～
      ‾‾教訓
        }
  cpattern{
    ‾‾教訓（
          （No‾A
           class 教訓
           タイトル ’‾B’
           ‾‾‾分類
           文書ファイルＩＤ   ‾ID
           キーワード  {‾‾‾キーワード}
              ）。。。
          }
  gpattern{
    ‾‾‾タイトル ((‾タイトル))、
    ‾‾‾分類     ((分類1   ‾分類1
                  分類2   ‾分類2))
    ‾‾‾キーワード    ((‾k1   ,‾k2)){,}。。。
          }
  facet   {
    ‾分類1       ([インジケータ、キュー、 ‥‥  ])
    ‾分類2       ([初期設定、エラー、    ‥‥  ])
    ‾k1          (instance(keyword))
    ‾k2          (instance(keyword))
        }
  sewing-methods{
    framemainte(){
     ptn(‾‾教訓,
           (”教訓Ｎｏ”，‾A，，”文書ファイルIDは”、‾ID,
              ‾‾‾タイトル，‾‾‾分類,”キーワードは、”、‾‾‾キーワード)
              ‾‾‾タイトル，(”タイトルは、”、‾タイトル)、
              ‾‾‾分類、    (”分類区分１は、”，‾分類1,
                            ”分類区分２は、”，‾分類2,
              ‾‾‾キーワード、(‾k1、”    ”、‾k2)
       )
    }
}
```

図５．１３　教訓フレームに対するＦＯＲＭ定義の例

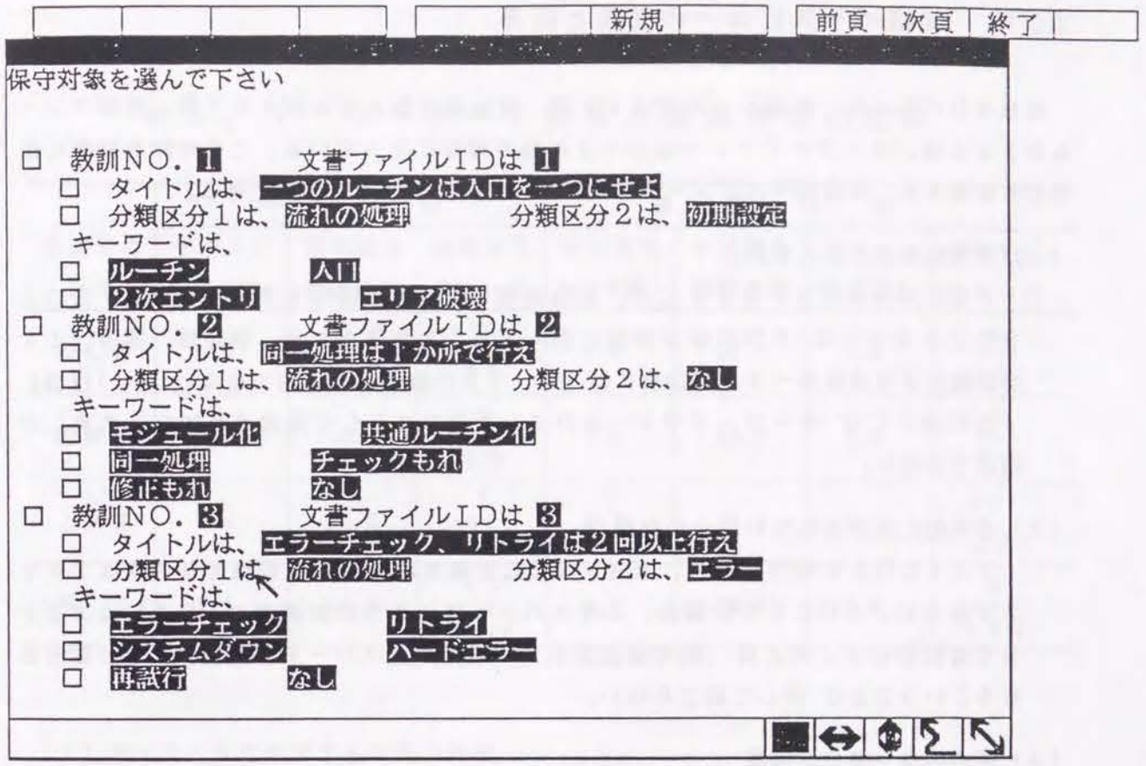


図５．１４　知識ベースビューを介した
教訓フレームのインタフェース画面例

同様に、教訓フレームの表現例である図５．６にも一定の形式があることが認められる。つまり、「教訓」と呼ぶ class フレームのインスタンスであることは固定であり、６ヵ所の可変部をもっている。その中でキーワード・スロットに入る値は「キーワード」class のインスタンス名が複数定義される。これらの構造定義とエキスパートとの対話法を定義したＦＯＲＭが図５．１３である。図５．１３を解釈したＴＡＩＬＯＲのインタフェースが図５．１４である。

これらの表示例から分かるように、ＴＡＩＬＯＲの示すインタフェースは、エキスパートシステム構築ツールの提供する知識モデルを隠している。そのため、ルール、フレームなどの知識の形式的表現に熟知していないエキスパートにも知識の保守が可能となると期待できる。実際、ＳＯＣＫＳの知識ベースの保守はこのインタフェースをエキスパートに提供することにより実施した。

５．５　知識ベースビューの効果と限界

現在ＳＯＣＫＳは、連想ルールが４１７個、関連度計算ルールが４０７個、教訓フレームが１６２個、キーワードフレームが４０４個の規模になっている。この知識を静的に保守した知見より、知識ベースビューの効用について次をあげることができる。

（１）不当な知識の混入を防止

ＦＯＲＭの中のファセットには、知識表現に関係する属性も含まれている。このようなファセットは 不当な値が知識に混入することを防止する。例えば、図５．１１の定義により連想ルールの確信度には［０，１］の数値しか入力できないし、「用語」「連想語」には キーワードフレームのインスタンスとして定義されている名称しか指定できない。

（２）基本的に変更されない部分の保護

ＴＡＩＬＯＲを使うかぎり、ＲＡＮＧＥに定義されていない知識セットにはアクセスできない。ＳＯＣＫＳの場合、エキスパートは、４つの知識セット以外にはアクセスできないので、例えば、順位決定表示ルールがエキスパートの誤操作により変更されるということは 決して起こらない。

（３）関心のない部分の隠蔽

エキスパートが 自分の関心のある部分にのみ 分かりやすい表現で 知識にアクセスできることが分かる。例えば、図５．３であげた検索フレームはインプリメントの便宜上、設けたものであるが、その存在を知る必要がない。

（４）知識の整理を支援

４．３．３で述べた機能（重複・冗長・矛盾の知識定義の検出機能）を使用することにより、条件部が同一あるいは包含関係にあるルールの検出、同一名称をもつルール、フレームの検出が可能である。

（５）知識の入力量を削減

この例では、キーボードからのキーイン数は、連想ルールの場合約２０％、教訓フレームの場合５５％に削減した。

ＳＯＣＫＳで実現した専門家モデルの有効性を評価するには、プログラミングノウハウの伝承が改善されたか、さらに その結果ソフトウェアの品質向上が実現できたかを分析する必要がある。現在ＳＯＣＫＳの利用を開始した時点であり、この測定は困難である。ここでは、従来のキーワード検索に対し、検索機能がどの程度改善されたかの評価を述べる。

初心者（入社１年～２年の新人）、中級者（入社３年～１０年の中堅）、上級者（入社１０年以上の管理者）の１６人の被験者に対し、７事例について評価を実施したところ表５．１を得た。

表５．１　キーワード検索と連想検索の比較

被験者の区分	キーワード検索			連想検索			
	平均入力キーワード数	平均適合教訓件数	検索不可のケース	平均連想キーワード数	平均適合教訓件数	平均検索順位	検索不可のケース
初心者	2.0	11.9	9/35	6.0	25.1	2.8	4/35
中級者	2.7	16.4	3/49	7.6	28.8	2.6	0/49
上級者	2.4	14.0	1/28	6.7	25.8	2.3	0/28
平　均	2.4	14.4	13/112	6.9	26.9	2.6	4/112

（１）キーワードを連想する知識の効用

単純なキーワード検索では検索漏れを生じる１３ケースのうち、キーワードの連想により、９ケースが検索可能となった。検索漏れの生じるケースの被験者は 初心者であり、入力キーワードの設定が不正確であった。

（２）教訓とキーワードを関連付ける知識の効用

従来のキーワード検索の適合事例件数は 平均１４．４件であるのに対し、本システムでは、キーワードの追加によって適合事例件数は平均２６．６件と増大する。しかし、その中で最も適合する教訓の検索順位は 平均２．６位であり、特に上級者では ２．３位であった。関連度計算ルールは 多数の検索結果の中から 目的とする教訓を迅速に得ることに有用であった。

これより、キーワードの連想で、検索対象が拡大され 検索もれを防いでいることが分かる。一方、教訓とキーワードの関連付けにより 検索対象を絞っており、大量の中から適切なものを選択する手間を省略していることが分かる。

有限の教訓に障害事例を対応付けるという意味で、ＳＯＣＫＳは 分類型エキスパートシステム［Clancey，1984］とみなすことができる。この種の応用は ビジネス分野に多く見られ、本報で述べた検索方式が適用可能なケースは多い。このように一つの専門家モデルの有効性が確認された場合、それは他の応用にも利用すべきと考える。

検索に関する専門家モデルを内蔵するＳＯＣＫＳは、エキスパートシステムの一つの類型を示すものである。ここに着目して、図５．８に示した４種類のビューを試行ビューとして使えるようにした [Yasunobu, et al. 1989]。この試行ビューを用いると、ＳＯＣＫＳが対象としたソフトウェア常識集以外の検索型エキスパートシステムを短期間でプロトタイピングすることができる。この場合、図５．３に示した検索フレームなどインプリメントの都合上のものは そのまま流用して、いくつかのエキスパートシステムを構築できた [Hashimoto, et al. 1988]。

一方、試行したエキスパートシステムに対する要求が 微妙にＳＯＣＫＳに要求されたものと異なる場合が予想される。このような要求は、試行により始めて気付かれることが少なくない。単純な可能性としては、検索対象の分類区分が２種類でなく、任意の種類とする要求が考えられる。複雑なものとしては、連想ルールとして一括して取り扱っていたものから類義語ルールを分ける、あるいは連想取り消しルールという考え方を導入するなどの要求があると思われる。

これら試行ビューが実用エキスパートシステムを開発する上で不適な場合、専門家モデルの修正が必要となり、それに従ってエキスパートシステムのアーキテクチュアの変更が必要となる。しかし、ナレッジエンジニア向けの知識モデル・レベルに戻り、システムを拡張することにより対応が可能である。

すなわち、試行環境で生成された知識ベースは 変更なしに使用できる場合もあれば、専門家モデルの修正を要する場合もある。修正を要する場合、一から設計するのでなく、試行結果として得られた知識ベースをもとに、知識モデル・レベルで知識を修正することができる。その結果、新たな保守のための知識ベースビューが作成され、それが汎用化されれば、新たな試行ビューができる。このように、提案した知識ベースビューは エキスパートシステムのアーキテクチュアのリサイクルを可能とする。

他方、提案した静的保守方式には、いくつかの限界がある。

中でも知識全体の検証に関する課題は重要である。個々のルールは 一見正しくても 全体として正当に問題を解けないことがあるが、提案方式は この問題の解決策を与えない。確実度、関連度などの決定についても 特に支援機能を有していない。

この提案した知識ベースビューの限界に関して、ＳＯＣＫＳでは、このシステム特有の機能として、何人かの被験者のキーワードを入力とし、連想するキーワード、検索した教訓の順位と関連度、および推論に要した時間をプリントアウトする機能を有している。これにより、ルールの過不足、関連度や確実度の補正作業を支援している。

５．６　まとめ

本章では、第４章で提案した知識ベースビューを適用しその有効性を検証するため、プログラミング・ノウハウを伝承するためのエキスパートシステムの構築について論じた。このエキスパートシステムＳＯＣＫＳはソフトウェア常識集と呼ばれる教訓集を検索するものである。

はじめに、本エキスパートシステムが必要とされるソフトウェア開発の現状について述べ、問題を解決する専門家モデルを示した。続いて、このモデルを実現するための知識表現について述べた。

ここで述べた検索方式では、とりあえずの知識で問題解決をしておき、問題解決の蓄積と共に知識を洗練化していくことが重要と考えた。これは、「開発に用いたエキスパートシステム構築ツールの知識表現やＳＯＣＫＳ全体の詳細仕様を熟知しているナレッジエンジニアが、いつまでもＳＯＣＫＳの保守に携わることはコスト的に現実的でない」と考えたためである。そのため、収集した知識を「継続的に保守される部分」と「基本的に変更されない部分」に分け、継続的に保守される部分について構造と属性を定義することにより、第４章で提案した知識ベースビューを適用した。

その結果、提案した知識ベースビューには 次の効用があり、ルール、フレームなどの知識表現に熟知していないエキスパートにも知識ベースの保守を支援した。

（１）知識の化読性が向上した。
（２）不当な知識の混入を防止した。
（３）基本的に変更されてはならない部分が隠蔽され、保護が実現された。
（４）関心のない部分を意識する必要がなくなった。
（５）知識の整理を支援した。
（６）知識の入力量を削減した。

検索に関する専門家モデルを内蔵するＳＯＣＫＳは、エキスパートシステムの一つの類型を示すものである。このような専門家モデルをもつエキスパートシステムの保守ビューは、一般に、試行ビューとしても利用できる。試行ビューで生成された知識ベースは 場合により専門家モデルの修正を要する。修正を要する場合、開発環境で、試行結果として得られた知識ベースをもとに修正する。その結果、新たな保守のための知識ベースビューが作成され、また、新たな試行ビューができる。

このように、提案した知識ベースビューは エキスパートシステムのアーキテクチュアのリサイクルを可能とする。本章で述べた検索のような汎用的な専門家モデルが見つかり、この種の類型化が複数でき、標準的なＦＯＲＭを見出すことにより、エキスパートの構築は 順次 容易なものになることが期待できる。

第６章　知識ベースビューによる知識ベースの動的保守

６．１　まえがき

本章は、第３章で導入した知識ベースビューの具体化の例として、動的に戦略知識を獲得していく方式に関するものである。ここで「動的」とは、推論実行時ということを意味し、戦略知識とは、ある時点で適用されるルールの候補が複数あるとき どのルールを選択するかに関する競合解消の知識である。

はじめに、ＭＥＡ（Means-Ends Analysis）と呼ぶ問題解決手法における領域知識と戦略知識について述べる。領域知識とは 対象世界の状態を変えるオペレータ（ルール）であり、戦略知識は どのオペレータ（あるいはどのゴール）を選択すべきか に関するメタレベルの知識である。

次に、従来 知られていた戦略知識だけを具備する専門家モデルでは、効率の良い探索ができないことを例を用いて示し、２種類の新たな戦略知識を組み込んだ専門家モデルを提案する。

一つの戦略は、ゴール・インタラクションが起こる特殊な状況において、本来適用されるべきオペレータの適用を見送るものである。この戦略は「急がば廻れ」と解釈できるものであり、ゴール・インタラクションが不可避の場合には、必ずしもスタック上にあるサブゴールを追跡することが全体の問題解決にとって最適ではないことに着目して得られる。

もう一つの戦略は、問題解決器（以下プロブレム・ソルバあるいは単にソルバとも呼ぶ）が 現在の状態を過去に解いたパターンと同一であるとみなしたとき、スタックしたゴールを 始めの一つを除いて 全てクリアするものである。この戦略は 最終的なゴールさえ満たせば、必ずしもスタック上の他のサブゴールを満たす必要がないことに着目して得られる。

続いて、提案した戦略知識を静的に獲得することの難しいことを示し、推論過程で戦略知識を保守するための知識ベースビューを提案する。元の専門家モデルの戦略知識同様、これらの探索戦略の知識は、プロブレム・ソルバの誤動作事例から 動的に獲得される。提案する戦略知識は、既に満たしたゴール、現在のゴールスタック、現在の状態、によって索引つけられる事例によって表現される。

最後に知識ベースビューを用いて獲得した戦略知識の効果を シミュレーション実験に基づいて定量的に示す。

６．２　領域知識と戦略知識を必要とする推論モデル

探索は エキスパートシステムの研究における古くからの課題である［Nilsson. 1980］。Means-Ends Analysis (MEA) [Fikes, et al. 1971][Newell, et al. 1972] は、計算機が取り扱いやすい 基本的な探索手法である。この一般的な手続きは、問題の状態を、逐次、目標状態に近づくように オペレータを選択することである。ＭＥＡを用いると、他の戦略を用いることなく問題を解けることが多い。しかしながら、ＭＥＡに基づく従来のプロブレムソルバは、組み合わせ爆発のために 現実的な時間では 小さな問題しか解くことができなかった。

探索の効率を改善するために、いくつかの手法が提案されている。例えば、オペレータ列で定義されるマクロオペレータ［Fikes, et al. 1971］は、現在の状態と目標状態の距離（解の長さ）を短くする。一方、複数のオペレータの候補から 一つのオペレータを選ぶ制御知識を用いる方法［Minton. 1988］［Mitchell, et al. 1983］は組み合わせ爆発の抑制に有効である。しかし、一般には、マクロオペレータは 組み合わせ爆発を増加させる欠点をもち、制御知識は 現在の状態から目標状態へ遷移する解の長さの短縮には 有効ではない。

近年、過去の問題解決の事例を再利用すること［Shank. 1982］が、組み合わせ爆発を抑制しつつ、解の長さの短縮に貢献すると期待されている。この考え方は事例ベース推論［Bradtke, et al. 1988］［Hammond. 1986］［Kolodner. 1985］［Ruby, et al. 1989］と呼ばれている。

本節では、従来のＭＥＡを専門家モデルとしてもつＰＲＯＤＩＧＹ［Minton. 1988］［Minton, et al. 1988］の探索戦略の限界を示し、事例ベース推論の観点から新たに２種類の探索戦略を提案する。これらは どちらもサブゴールの操作を含んでいる。その後、それらをＰＲＯＤＩＧＹにインプリメントする方式について論じる。

６．２．１　ＭＥＡに基づくシステムＰＲＯＤＩＧＹの概要

以下の議論の準備として、はじめにＰＲＯＤＩＧＹの専門家モデルについて述べる。簡単にいうと、ＰＲＯＤＩＧＹ の探索は

（ａ）領域知識（オペレータ）

（ｂ）制御ルール（戦略）

（ｃ）問題（初期状態と目標状態）

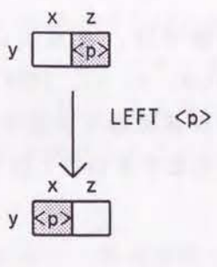
を与えられたときに、初期状態から目標状態（以下ゴールと呼ぶ）へのオペレータ列を求めるものである。ＰＲＯＤＩＧＹの知識表現の形式的定義の概要については付録に示している。

領域知識は 状態を遷移するための知識である。それぞれのオペレータは それを適用する前提条件と前提が満たされたときの状態の遷移を指定する。格子上の位置を左に移動するためのオペレータの例を図６．１に示す。以下、格子上を移動するエイトパズルを事例に知識表現の例を説明する。

制御ルールは 競合解消に関するものであり、複数のオペレータが適用可能であるときの選択方法、複数のゴールを満足させるとき満足させるゴールの優先順序などを指定する。例えば、図６．２では 与えられたゴール（右下から左上に移動）に対して ２つのオペレータが適用可能であるが、左下がブランクであることに着目すると、案２の方が より短い解を得る。このような場合、図６．３のように 制御ルールが 指定される。

初期状態と目標状態を定義する問題の例を 図６．４に示す。

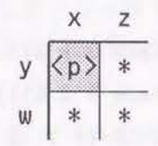
```
(LEFT    (parms    (<p>))
         (preconds (and (hnei <x> <z>) (iv <y>)
                        (place <z> <y> <p>)
                        (blank <x> <y>)))
         (effects  ((del (blank <x> <y>))
                    (del (place <z> <y> <p>))
                    (add (place <x> <y> <p>))
                    (add (blank <z> <y>)))))
```



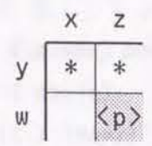
ここで、述語 (place <x> <y> <p>) は、 <p> が、座標 (<x> <y>) にあるとき真で、
述語 (blank <x> <y>) は、座標 (<x> <y>) がスペースのとき真であるとする。
述語 (hnei <x> <z>) は <z> が <x> の右であるとき真である.

図６．１　ＰＲＯＤＩＧＹにおけるオペレータの例

目標状態



現在状態

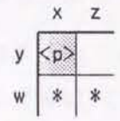


但し、'*' は何でも良いことを示す.

このとき、ゴールを満たす上で２つの代替案がある;

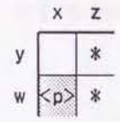
案１：

右の状態から左に動かす



案２：

右の状態から上に動かす



現在状態では、案２が好ましい

図６．２　オペレータ適用における競合の例

```
(SELECT-UP (lhs (and (current-node <node>)
                     (current-goal <node> (place <x> <y> <p>))
                     (candidate-op <node> LEFT)
                     (candidate-op <node> UP)
                     (known <node> (and (hnei <x> <z>)
                                        (vnei <y> <w>)
                                        (blank <x> <w>)
                                        (place <z> <w> <p>)))))
           (rhs (select operator UP)))
```

図６．３　ＰＲＯＤＩＧＹにおける制御ルールの例

```
(load-goal       '(and (place 1 1 1) (place 2 1 2) (place 3 1 3)
                       (place 1 2 4) (place 2 2 5) (place 3 2 6)))
(load-start-state '((vnei 1 2) (vnei 2 3) (hnei 1 2) (hnei 2 3)
                    (ih 1) (ih 2) (ih 3) (iv 1) (iv 2) (iv 3)
                    (place 1 1 8) (place 2 1 5) (place 3 1 2)
                    (place 1 2 7) (place 2 2 4) (place 3 2 1)
                    (place 1 3 6) (place 2 3 3) (blank 3 3)))
```

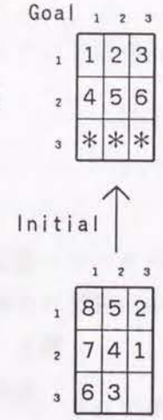


図６．４　ＰＲＯＤＩＧＹにおける問題の例

問題は初期状態と目標状態の組で表される

６．２．２　ＰＲＯＤＩＧＹの探索アルゴリズム

ＰＲＯＤＩＧＹは、問題を解くために 探索中にツリーを生成する。この探索ツリーは、初期状態とゴールのペアからなる一つのノードを頂点とし、次の２ステップを繰り返すことにより 枝を伸長していく。

１．決定フェーズ

探索ツリーの 一つのノード、そのノードで注目するゴール、そのゴールを実現するためのオペレータ、そのオペレータに対するバインディングの組を決定する。

２．展開フェーズ

オペレータが現在の状態に対して適用可能な場合、それを適用する。適用できない場合、満足しない条件をサブゴールとする。新しいノードは 次のように作る：

（ａ）オペレータが適用可能な場合：

・サブゴールは 親ノードを生成した条件に依存して決定する。
・ゴールスタックは 親ノードからひとつゴールを除去したものとする。
・状態は 親ノードの状態にオペレータを適用したものとする。
・新しく生成した状態が 探索ツリーの先祖のノードと一致しないかを調べる。一致した場合、ループに陥るので その子ノードの探索を停止する。

（ｂ）サブゴールを作る必要がある場合:

・選択したオペレータの中で、満足しない条件を 新しいサブゴールとする。
・ゴールスタックは 親ノードのゴールスタックに 新しいサブゴールを付加する。
・状態は 親ノードと同一とする。
・付加したゴールが ゴールスタック上にないかを調べる。存在した場合、ループに陥るので、その子ノードの探索を停止する。

決定フェーズにおける制御ルールは、組み合わせ爆発を抑制したり、解の質の向上を実現したり、ソルバの探索方向を誘導したりする上で非常に重要である。

制御ルールの獲得を容易にするため、ＰＲＯＤＩＧＹはＥＢＬ (explanation-based learning)[Minton, 1988] [Mitchell, et al. 1986] と呼ぶ学習コンポーネントを持っている。ＥＢＬは、一つの例である問題解決のオペレータ列を解析することにより、何故、そこでの決定が適当であるかの説明を生成し、それを制御ルールに変換する。

６．２．３　ＰＲＯＤＩＧＹの限界

しかしながら、以上で 示した探索アルゴリズムには次の問題がある。：

(1) ゴールスタックから選択されるゴールを達成するオペレータを優先的に適用。

しかし、あるゴールを達成する途上に、既達成のゴールに干渉してしまうことがある。このようなゴール・インタラクションが起こる場合、ゴール・スタック上のゴールを優先的に達成することが全体の問題解決になるという仮定は 必ずしも正しくない。

(2) スタック上の サブゴールを達成するオペレータの包含は必須。

しかし、本来の問題は、与えられたゴールを達成することであり、サブゴールを達成することではない。初期状態とゴールの距離が大きい場合、スタックに多くのサブゴールを積むことになるが、探索ツリーが深くなった場合、ＰＲＯＤＩＧＹが前提としている「サブゴールを達成することが 問題を解く必要条件である」という仮定は必ずしも正しくない。

これらの問題は、解を求められなくするものではない。場面特有の制御ルールにより解けたり、最適解でない質の劣る解を見つけたりする。しかし、より優れた解を 特殊な制御ルールを定義することなく求めることが望ましい。

以下では、これらの２つの問題の具体例を示し、解決する方法について 考察する。

6．2．4　PRODIGYの限界を解決する戦略知識

最初に 問題(1)について考察する。元のPRODIGYでは 複数のゴールを達成しなければならないとき、特定のゴールを選択、拒絶、選好する制御ルールを記述する。すなわち、制御ルールは ゴールインタラクションを避けるため、いかなるサブゴールを優先的に選択すべきかを定義する。

しかし、問題は 単純ではない。エイトパズルを用いてこれを説明していく。

図6．5で表わされる初期状態とゴールからなる問題を考える。ここでは、初期状態で既に2つのゴール（place 1 1 1）と（place 2 1 2）が満たされている。このとき ソルバは、次のようにサブゴールをゴールスタックに積む：

S1 : (place 3 1 3),
S2 : (blank 3 1),
S3 : (blank 2 1),

ここで（place A B C）は"C"を 座標（A B）におくことを示し、(blank A B) は座標（A B）がスペースであることを示す。

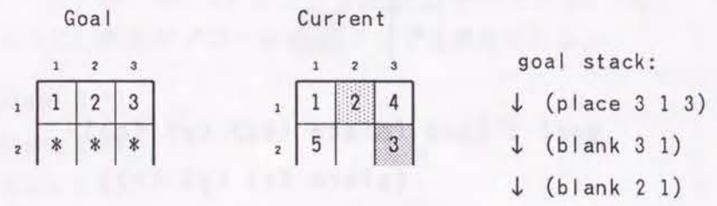
このとき、(blank 2 1) を満たすために、2を下げるオペレータ（DOWN 2）が適用される。MEAに従うと、このオペレータの適用を避けることができない。この場合、2が既にゴールの位置にあるので、ゴールインタラクションが起こる。

しかし、この問題の最短プロセスは 次であり、(DOWN 2)を含まない：(RIGHT 5), (DOWN 1), (LEFT 2), (LEFT 4), (UP 3), (RIGHT 5), (DOWN 4), (RIGHT 2), (UP 1)。このように ゴール・インタラクションのために 最適プロセスが見損じられることがある。

一般に、エイトパズルを解く場合、一つのゴールに固執することは得策ではない。後で再度満たすようにしなければならないかもしれない他のゴールのことを勘案しながら、現在のゴールを達成しようとする。すなわち、現在の状態とゴールを見比べながら、必ずしもMEAの意味では候補とならないオペレータを利用する。

従って、プロブレムソルバは、状態空間に機先を制す（ forestall ）ことができれば、より知的となる。状態空間フォーストールとは、ゴール・インタラクションが起こる状態を認識し、それを起こすオペレータの利用を避け、他のオペレータを挿入することを言う。この戦略を具備するために、プロブレムソルバには フォーストール・オペレータと呼ぶ知識を導入する。各フォーストール・オペレータは 適用するための条件部と いかなるオペレータを利用するかの結論部をもつ。条件部には 既に達成したゴールとゴールスタック上のサブゴールを含む。この戦略は ことわざで言う「急がば廻れ」に対応する。

Goal : (and (place 1 1 1) (place 2 1 2) (place 3 1 3))

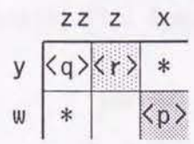


ゴール（place 3 1 3）を満たそうとすると
(DOWN 2) が選択される．しかし、問題全体を見ると
ゴールスタックをクリアしない（RIGHT 5）を適用すべきである．

図6．5　フォーストールオペレータを必要とする
ゴールインタラクションの例

フォーストール・オペレータは、事例を一般化することにより 獲得される。現在のゴールと現在の状態とこの時点で既に達成しているゴールに注目して、図６．５の状態は定数を変数化することにより図６．６のように 一般化できる、このとき フォーストール・オペレータは図６．７のように定義できる。

一般に 人間は 前向き推論と後向き推論を 組み合わせて問題解決を行なっている[Anderson. 1983]。フォーストール・オペレータは 後向き推論のプロブレムソルバに 前向き推論の機構を統合したとみなすこともできる。




```
goal : (and (place <zz> <y> <q>)
            (place <z> <y> <r>)
            (place <x> <y> <p>)
       )
```

図６．６　図６．５の状態を一般化した状態

```
(FORESTALL (if (and (current-node <node>)
                    (current-goal <node> (place <x> <y> <p>))
                    (achieved-goal <node> (and (place <zz> <y> <q>)
                                               (place <z> <y> <r>)))
                    (known <node> (and (vnei <y> <w>)
                                       (hnei <zz> <z>)
                                       (hnei <z> <x>)
                                       (blank <z> <w>)
                                       (place <x> <w> <p>)))))
           (then (force RIGHT))).
```

図６．７　フォーストールオペレータの例

次に 問題(2)について 考察する。図６．８(a)に示す初期状態とゴール (place 1 1 1) に対しプロブレムソルバは 次のように サブゴールをスタックに積む。:

S1 : (place 1 1 1),
S2 : (place 2 1 1),
S3 : (place 3 1 1),
S4 : (place 3 2 1),
S5 : (blank 3 2),
S6 : (blank 2 2),
S7 : (blank 1 2).

このとき (UP 6) が 適用可能となる(図６．８(b))。さらに (LEFT 5), (LEFT 4), (UP 1) が適用され、 S7, S6, S5, S4 がスタックから 除去される。(図６．８(c))次に図６．８(d)に示すように、次のサブゴールが スタック上に積まれる。

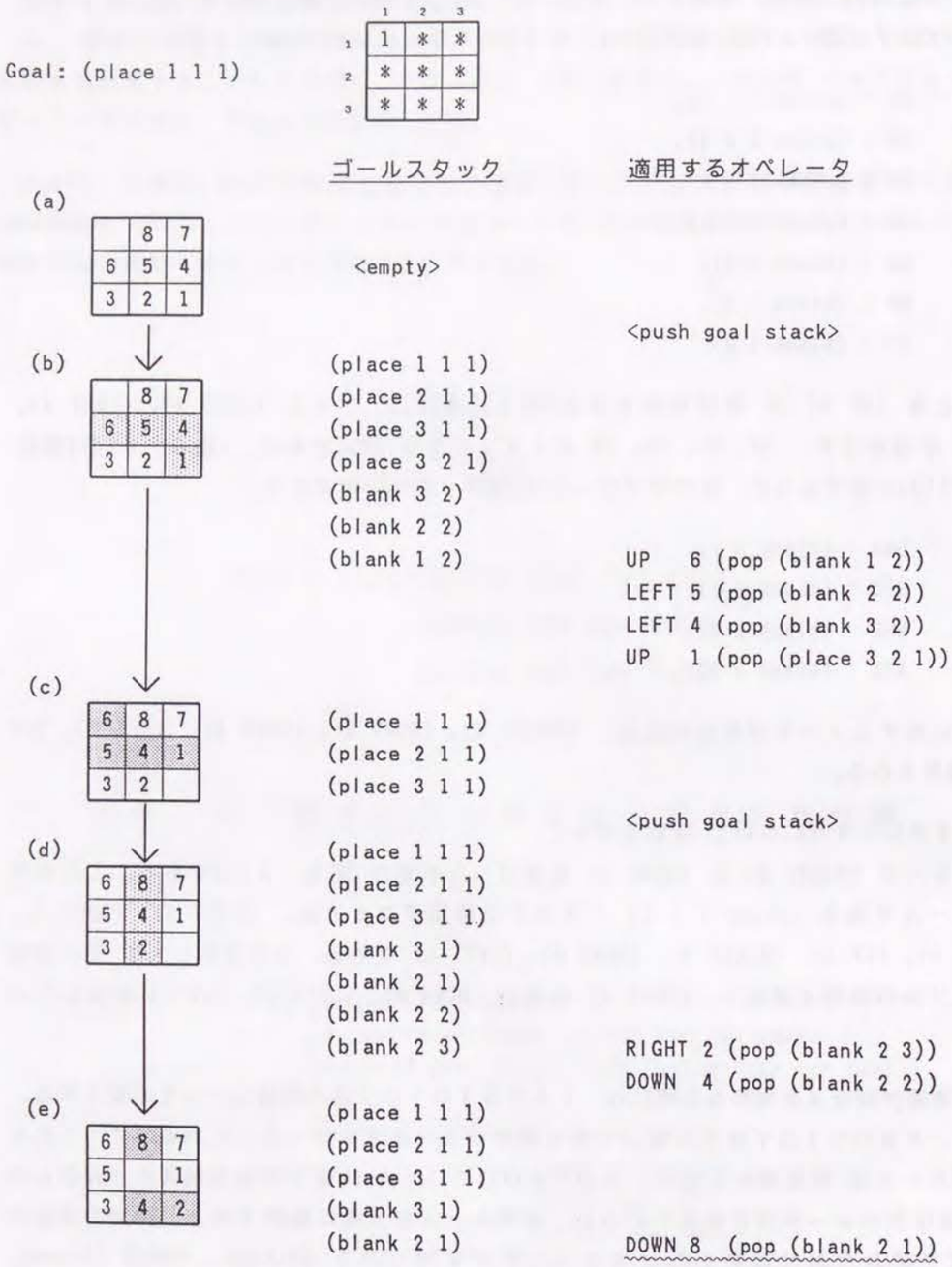
S42 : (blank 3 1),
S52 : (blank 2 1),
S62 : (blank 2 2),
S72 : (blank 2 3).

S72 に対するノードが作られた後、 (RIGHT 2), (DOWN 4), (DOWN 8), (LEFT 7), (UP 1)が 適用される。

このオペレータ列について 検討を行なう。

オペレータ (RIGHT 2) と (DOWN 4) を適用した状態が 図６．８(e)である。この状態からゴールである (place 1 1 1) へ到達する最短プロセスは、 (LEFT 1), (DOWN 7), (RIGHT 8), (UP 1), (RIGHT 5), (DOWN 6), (LEFT 1) である。このプロセスは 先のプロブレムソルバの解と異なり、(DOWN 4) の後の (DOWN 8), (LEFT 7), (UP 1) を含んでいない。

この最適プロセスを求めるためには、ＰＲＯＤＩＧＹは大量の制御ルールを必要とする。しかし、ＰＲＯＤＩＧＹはこの場面で多くのサブゴールをスタックしているので、これらの制御ルールは 場面特有となる。元のＰＲＯＤＩＧＹは 大量の特殊な制御ルールなしでは 最適なオペレータ列を求められない。制御ルールを大量に獲得すると、探索の決定フェーズで組み合わせ爆発を起こすという弊害を持つので [Minton. 1988a] [Minton. 1988b]、場面特有の制御ルールを定義するのではなく、別のメカニズムを探すべきと考える。



(e) の状態でサブゴール (blank 2 1) を満たすため (DOWN 8) が適用される。しかし、最終ゴール (place 1 1 1) を満たすには、(LEFT 1) の方を採用すべきである.

図６．８　ゴールスタックのクリアを必要とする例

一般に、我々が問題を解くときには、ゴールスタックに固執しない。しばしば、ＭＥＡに近いことを行なっているが、ゴール・インタラクションの存在に気付いたり、最終的なゴールへの近道の存在に気付いたときは、ＭＥＡには固執しない。すなわち、現在の状態空間を分析した結果、我々はゴールを達成する上で満たす必要のないスタック上のサブゴールを 意図的に クリアする。

それ故に、プロブレムソルバも スタック上の不必要なサブゴールをクリアする能力を持てば、さらに知的になる。実際、より良い解を得るためには、図６．８の（ｃ）においてこれを初期状態をみなし、スタック上のサブゴールを（place 1 1 1）を除いて全てクリアすべきである。この戦略を 「一から始めよ」と呼ぶ。

この戦略は、オペレータ列を一つにまとめて実行するマクロオペレータ [Fikes, et al. 1971] に類似している。しかし、この戦略は マクロオペレータとその適用法が異なる。マクロオペレータは あくまで オペレータの一種であり、ソルバはゴールスタック変更することなく、マクロオペレータを適用できる状態を探す。一方、提案した戦略はあらかじめ登録しておいたパターンの検出を試み、検出されると 問題解決を続ける代わりに強制的にゴール・スタックをクリアする。

大量の制御ルールが定義され 新しいルールが次から次へと追加されるとき、ルール間の無矛盾性を保つことは 必ずしも容易ではない。それに対し、ここで述べたパターンについて言えば、それらのパターンは互いに干渉しないので、一貫性に関して 特に 配慮する必要がない。それぞれのパターンは追っているゴールと状態によって索引付けられる。上の例から一般化されたパターンと それを対応する知識に表現したものを 図６．９、図６．１０に示す。

これら２種類の戦略知識は 先に示した展開フェーズを拡張することにより インプリメントできる。拡張したアルゴリズムを図６．１１に示す。

	zz	z	x
y	*	*	*
w	*		<p>

目標状態 : (place <zz> <y> <p>)

図６．９　図６．８(e)を一般化した状態

```
(GOALCLEAR      (if (and (current-node <node>)
                         (current-goal <node> (place <zz> <y> <p>))
                         (on-goal-stack <node> <extra subgoal>)
                         (known <node>  (and (vnei <y> <w>)
                                             (hnei <zz> <z>)
                                             (hnei <z> <x>)
                                             (blank <z> <w>)
                         (place <x> <w> <p>)))))
                (then clear-goalstack))).
```

図６．１０　ゴールスタックをクリアする戦略知識の記述例

オペレータが現在の状態に対して、適用可能な場合、
　その状態でフォーストールオペレータに適用可能なものがあれば、適用、
　　なければ、オペレータを適用。

適用できない場合、サブゴールを作成。
　このとき、ノードは次のように更新。
　（ａ）オペレータを適用する場合、
　・ サブゴールは、親ノードを生成した条件に依存して決定。
　・ オペレータ適用後の状態が、ゴールスタックをクリアするパターンのとき、
　　○ スタック上に積まれた最初のサブゴール以外の全てのサブゴールを
　　クリア。
　　パターンでないときは、
　　○ サブゴールは、親ノードを生成した条件に依存して決定。
　　○ ゴールスタックは、親ノードからひとつゴールを取り除いたものとする。
　　○ 新しい状態がループに陥らないかをチェック。
　（ｂ） サブゴールが必要な場合、６．２．２の（b）に従う。
　（ｃ） フォーストールオペレータが適用可能な場合、
　・ サブゴールとゴールスタックは変化させない。
　・ 新しい状態を該当するフォーストールオペレータを適用することにより
　　作成。

図６．１１　戦略知識を統合した探索アルゴリズム

６．３　戦略知識を動的に獲得するための知識ベースビュー

６．３．１　知識獲得の一般的アルゴリズム

前節で述べたように 探索を目的とするプロブレムソルバの能力を向上するには、次の戦略知識を獲得する必要がある。

（１）制御ルール
（２）フォーストール・オペレータ
（３）サブゴールをクリアするためのパターン

これらの戦略知識の条件部はゴールスタックの情報を含むので、推論と独立に獲得することが困難である。従って、初期知識として領域知識と既得の戦略知識を用いた問題解決を行い、ソルバに誤動作があるとき、それに気付くことにより、ゴールスタックなどの情報を参照し獲得すべきであると考える。

ソルバが 適用するオペレータ案を提示したとき、そのオペレータが 我々の予想と異なる場合がある。異なっていれば、戦略知識を動的に獲得できる可能性がある。知識獲得のコンポーネントを 図６．１２、フローチャートを 図６．１３に示す。ここで、図６．１２に示すエディタは、そのときの状態空間の参照により、戦略知識の条件部を利用者に提示できることに注意しておく。

ここでは、解の大半は ＭＥＡに基づいて得られると仮定しているので、ソルバは 特に戦略知識を具備していなくても 必要な探索を行うし、小規模な問題であれば、時間をかけずに解くことができる。しかし、問題が複雑になるとともに戦略知識なしでは探索空間が大きくなり、合理的な時間で解くことができないので、順次、戦略知識を獲得することが好ましい。

ソルバの候補と我々の期待が異なったとき、次のアルゴリズムで戦略知識を獲得する:

[Step 1]

我々が適用すべきと考えるオペレータが その状況でＭＥＡの意味で候補の一つであるかどうかを調べる。

これは、ユーザ・インタフェース・ファシリティを用いて探索ツリーを調べることにより分かる [8]。この機能は サーチ・トレースを分析する機能を具備しており、ユーザが システムの動きを評価したり 強制的に変更したりすることができる。

[Step 2]

オペレータが 候補の一つであれば、考えられる制御ルールが場面特有（すなわちコストエフェクティブ[Minton, 1988b]）であるかどうかを 検討する。

特殊な場面でしか使用されない制御ルールを具備しているとその評価にかかるコストの方が大きく、制御ルールによる効果ではそれを相殺できないという問題がある。

従って、頻繁に使用される制御ルール以外を獲得することは得策ではない。場面特有というのは主観的な表現であるが、例えば、「ソルバが５個以上のサブゴールをスタックに積んでいる」というように定義される。

[2.1] 制御ルールが 場面特有でなければ、それを獲得する。
そのときの状態を一般化してルールの条件部を記述する。

[2.2] 制御ルールが 場面特有であれば、
最初に積んだサブゴール以外をクリアできると仮定する。
[2.2.1] ソルバがサブゴールをクリアした状態から問題の続きを解き、より良い解を得られるならば 仮定は検証されたとし、先の状態を一般化し、サブゴールをクリアできるパターンとして システムに記憶する。一般化するために、現在のゴールや現在の状態に注目し、変数化できる部分を抽出する。
[2.2.2] 良い解を得られない場合は、この場で 戦略知識を獲得することは難しいとし、諦める。

[Step 3]

期待するオペレータが ＭＥＡの意味で候補以外であり、そのソルバの採用しようとしているオペレータが ゴールインタラクションを起こすならば、状態空間に機先を制しようとしているのではないか と仮定する。

[3.1] ソルバに機先を制すためのオペレータを適用し、その後の状態から問題の続きを解かさせ、より良い解を得られるならば、仮定は検証されたとし、我々の期待をフォーストール・オペレータとする。ここで、フォーストール・オペレータの左辺は 先の状態を一般化したものであり、右辺は 代用したオペレータである。フォーストールオペレータを一般化するため、現在のゴールと現在の状態と既に満足したゴールとに注目する。

[3.2] 良い解を得られない場合は、この場で 戦略知識を獲得することは難しいとし、諦める。

[2.1], [2.2.1] と [3.1]において、一般化の手法が課題となる。一般化とは知識の適用範囲を広めるために条件部を緩和することであり、述語における定数の変数化と一部の述語そのものの削除を含む。これまでの経験から 事例から一般化することにより戦略知識を獲得することは、発見的に戦略知識を獲得するより容易であるが、過剰一般化を避ける方法は 今後の課題である。

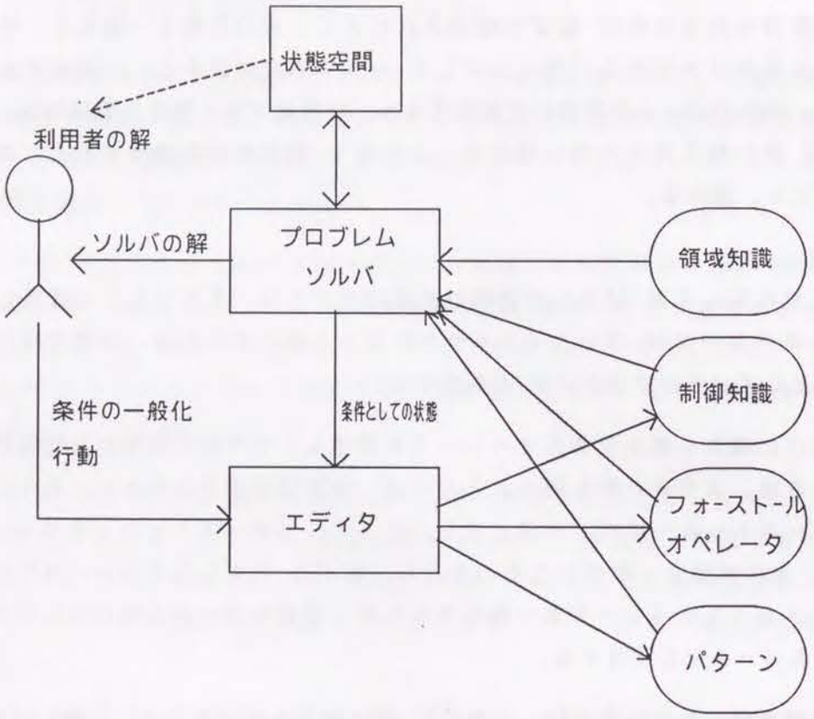
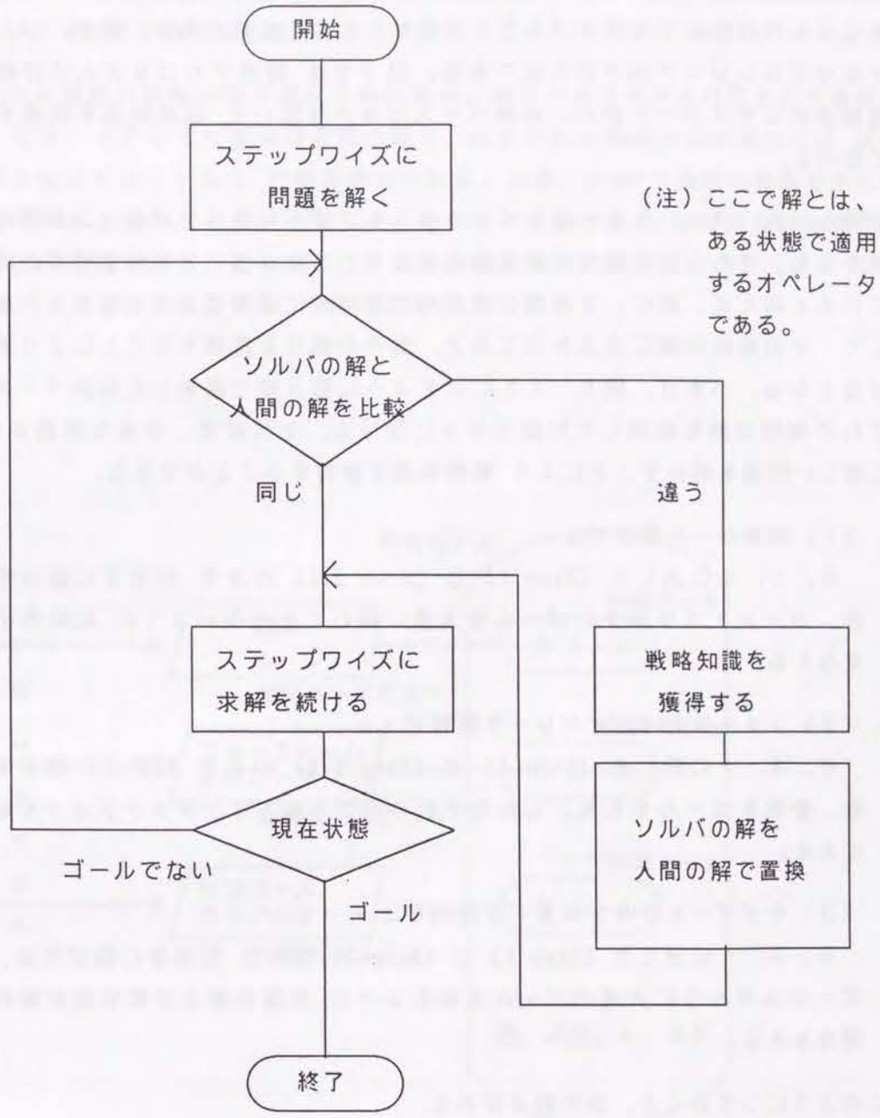


図６．１２　戦略知識を獲得するための機能構成



（注）ここで解とは、ある状態で適用するオペレータである。

図６．１３　戦略知識を獲得するアルゴリズム

６．３．２　知識ベースビューによる逐次知識獲得方法

先に述べたアルゴリズムによる知識獲得は、静的な知識獲得より負担が少であるが、利用者はソルバの探索アルゴリズムを十分熟知しておく必要がある。即ち、ソルバを開発したナレッジエンジニア向けの方法である。以下では 探索アルゴリズムの詳細を知らない問題解決のエキスパートから、知識ベースビューを用いて 戦略知識を獲得する方法について述べる。

制御ルール、フォーストールオペレータ、サブゴールクリアパターンが獲得できる状況は異なるし、それらの有効性の検証法も異なる。これがここでの知識獲得の問題を難しくしていると考える。逆に、３種類の戦略知識を同時に獲得するのではなく、ある側面に注目して、その戦略知識に焦点を当てると、動作の誤りを指摘することにより獲得することが可能となる。つまり、図６．１４に示すように第３章で提案した知識ベースビューをそれぞれの戦略知識を格納した知識セットに設ける。その結果、簡単な問題から始めて 徐々に難しい問題を解かすことにより 戦略知識を獲得することができる。

（１）制御ルール獲得ビュー

６．３．１に示した［Step 1］と［Step 2.1］のみを 利用者に提示する。この場合、ゴール・スタックにゴールを大量に積むことがないように 比較的小規模の問題を与える。

（２）フォーストールオペレータ獲得ビュー

６．３．１に示した［Step 1］と［Step 2.2］のみを 利用者に提示する。この場合、複数をゴールをもち、しかもそれらがゴール・インタラクションを起こす問題を与える。

（３）サブゴールクリアパターン獲得ビュー

６．３．１に示した［Step 1］と［Step 3］のみを 利用者に提示する。この場合、ゴールスタックに大量のゴールを積むように 初期状態と目標状態が離れた複雑な問題を与える。

このようにしておくと、次の利点がある。

（ａ）ソルバに熟知している場合には、６．３．１のアルゴリズムで３種類の知識を並行して獲得できる。これは ナレッジエンジニアが開発環境で知識ベースを構築することに対応する。

（ｂ）ソルバに熟知していない場合には、ある側面に注目して戦略知識の獲得ができる。獲得のタイミングや他にいかなる戦略知識があるかを意識することなく、作業を進められる。たとえ、戦略知識がなくても ソルバはそれなりの動作を行う。これはエキスパートが保守環境で知識ベースを洗練化していくことに対応する。

（ｃ）領域知識（オペレータ）が再利用可能な分野へは、上記 戦略知識の種類が十分との前提のもとで試行ができる。これは、エキスパートが 試行環境で知識ベースをプロトタイピングすることに対応する。

（ｄ）新たな種類の戦略知識が発見された場合、統合アルゴリズムはますます複雑になるが、ビューを介して知識獲得を行う限り、それぞれの戦略知識の獲得には 他の戦略知識を気にすることなく 作業を進められる。実際、今回 ２種類の戦略知識の獲得アルゴリズムを統合したが、上記（１）のビューを使う限り 従来通りの知識獲得を行うことが可能である。統合アルゴリズムの改良は、ナレッジエンジニアが 開発環境で 専門家モデルを修正することに対応する。

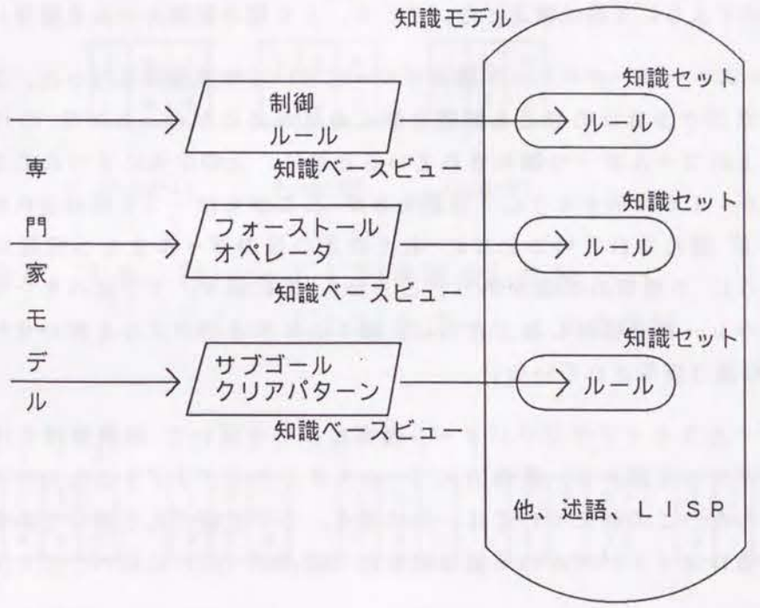


図６．１４　知識ベースビューによる戦略知識の獲得

６．４　エイトパズルへの適用例

提案方法を エイトパズルに適用して 戦略知識を獲得した。エイトパズルの特徴は次のとおりであり、戦略知識の獲得を考察するには 好適なものである：

（１）領域知識が単純（４つのオペレータ：LEFT, RIGHT, UP, DOWN)、
（２）人間にとって学習が容易、
（３）探索空間が大きい、
（４）探索時間や解に関し、最適なものを見つけることが困難、
（５）戦略知識を静的に獲得することが困難。

始めに、制御ルール獲得ビューにより 制御ルールを獲得した。場面特有の状態（サブゴールを多くゴールスタックに積む状態）での獲得を避けるため、縮小した ２×２のパズルを解いた。その結果、これらの問題に対して 最適解を得るように ６個の制御ルールを獲得した。続いて ３×２ のパズルについて、左上にゴール（place １ １ 1）をもつ問題を作り、知識獲得を行なった。２５種類の問題が 潜在的にあるが、ここで図６．１５に示すように５題に限定した。ここで、１６個の制御ルールを獲得した。

次に、フォーストール・オペレータ獲得ビューを用いて知識獲得を行った。ここでは、ゴール・インタラクションの起こる問題を解く必要があるから、３×２ のパズルを対象とした。左上のゴールが一つ満たされているときに、上の中央に２つめのゴールを持つ問題は、図６．１６に示すように７種類あるが ここからは一つも獲得されなかった。２つのゴールが 満たされているときに 右上の３つ目のゴールをもつ問題は、図６．１７に示すように ５種類の問題があり、ここから 先に図６．７で述べた一つのフォーストール・オペレータを獲得した。さらに、種々の３×３のパズルを解かせたが、今のところ、他の知識は獲得されていない。

その後、ゴールスタッククリアパターン獲得ビューを用いて 知識獲得を行った。種々の３×３のパズルを解かせ、最終的にゴールスタックをクリアするための１４個のパターンを獲得した。この例については、先に図６．１０で述べたとおりである。獲得したパターンの数はエイトパズルの可能な状態数 362,880 (9!) に比べ、ごく少ないものである。

以上の知識獲得を行なった後 ４つの問題を
（１）エイトパズルに準最適解を得ると期待できる人間
（但し ＭＥＡに従わない）
（２）制御ルールのみを具備するソルバ
（３）（２）に加え 提案した戦略知識を具備したソルバ
に解かした。シミュレーション結果を表６．１に示す。

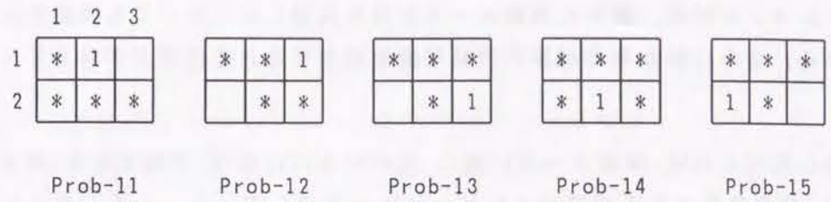


図６．１５　(place １ １ 1)をゴールとする状態

(注)潜在的には２５種類の状態が存在する

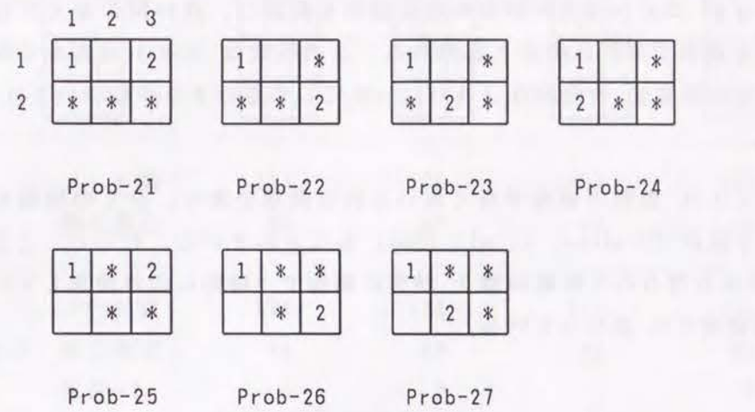


図６．１６　(place １ １ 1)を達成した後
(place ２ １ 2)をゴールとする状態

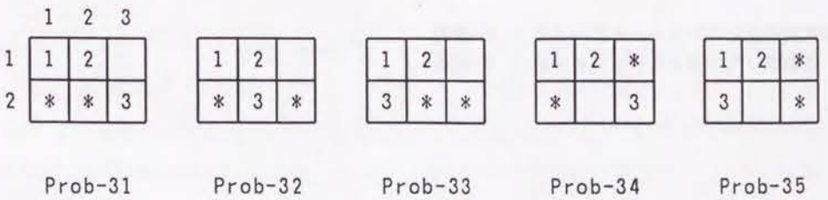


図６．１７　(place １ １ 1)と(place ２ １ 2)を達成した後
(place ３ １ 3)をゴールとする状態

＊ は任意の値で良いことを示す。

シミュレーションの結果、確かに制御ルールだけを具備したソルバでも問題解決できることが分かる。さらに新たな２種類の戦略知識を利用するとその能力が向上されることも分かる。

まず、統合したソルバは 探索ノードに関し 元のソルバに比べ 平均２３％ 減少している。これは 探索効率の向上が実現されていることを示している。一方、オペレータ列を平均２７％ 短くしており、これより 解の質の向上を実現していることが分かる。従って、戦略知識の動的な獲得は定量的にも有効であることが確かめられた。

しかし、統合したソルバといえども、解の長さが人間の２倍になることがある。これは、人間だけが エイトパズル特有の近接関係を認識し、良い関係がくずれないようにオペレータを選択できるためだと思われる。この戦略は エイトパズルでは極めて有効であるが、この戦略を 一般的なソルバにいかにして統合するかについては 今のところ不明である。

エイトパズルは 探索の研究で良く用いられる問題であり、全ての問題を事例ベースで解くという試み [Bradtke, et al. 1988] もなされている。しかし、ここでのアプローチは 事例から得られた戦略知識を 特別の場面で一時的にのみ使用しているという意味で 従来の研究とは 異なっている。

表６．１　知識ベースビューを介して獲得した戦略知識の効果

		目標状態											
		1 2 3		1 4 7		8 7		2 4 6		5 6 7	
		4 5 6		2 5 8		6 5 4		8 1 3		4 1 8	
		* * *		3 6		3 2 1		5 7		3 2	
PRODIGY	ノード数		157		198		122		217		
	解の長さ		66		80		54		88		
新たな	ノード数		114		152		122		144		
戦略をもつ	解の長さ		44		65		48		52		
ソルバ	回数-１		0		2		1		0		
	回数-２		3		5		5		3		
人間	解の長さ		42		26		38		26		

回数-１ ： フォーストールオペレータの使用回数.
回数-２ ： ゴールスタックをクリアした回数.

６．５　まとめ

本章では、第３章の構想に基づき、知識ベースを動的に保守するための方式について論じた。

まず、はじめに従来知られていた戦略知識では、質の良い解を効率良く求められないことを示し、新たな２種類の戦略知識を示した。ここでは、ＭＥＡにおけるゴール・インタラクションにまつわる欠点の存在が 研究の出発点となっている。この欠点を克服する戦略の一つは 状態空間に機先を制するものであり、もう一つは ゴールスタックを強制的にクリアするものである。提案した新たな戦略知識にある基本原理は、「ＭＥＡで問題を解くときに瞬間的に適用すると有効な戦略がある」ということである。この原理は 理論的に導いたものでなく、経験から得たものである。

続いて、これらの新しい戦略知識を問題解決時に獲得する方式について論じた。この方式は、

・最低限の領域知識で問題解決をさせておき、

・システムの動作に誤りがあると気付いたときに、そのときの状態を一般化することにより戦略知識を獲得する

ものである。インプリメントは 既存のソルバの展開フェーズを拡張することにより可能である。

アルゴリズム的には 種々の知識を同時に獲得できるが、与えられた問題の複雑さによって

・ある側面の戦略知識に焦点を当てたほうが獲得しやすいこと、

・そのとき、知識ベースビューという概念を導入することが有効であること、

を論じた。知識ベースビューにより、知識ベースのある一面に焦点を当てていると、利用者は 他の知識の存在を意識することなく、知識獲得に集中することが可能である。

提案した方式をエイトパズルの問題に適用し、実際にシミュレーションすることにより、従来の方式に比べ、探索ノードを２３％削減し、解の長さを２７％短くなることを確認した。戦略知識については 新たな種類のものが 今後 発見され ますます多様になる可能性があるが、このときも知識ベースビューが、ある一面の戦略知識に焦点を当てて獲得することを可能にする。実験に用いた問題は４種しかなく、定量的に提案したアプローチを評価するには少ないかも知れないが、今後 より複雑な割り当て問題やスケジューリング問題などの実問題に適用して、アイデアの有効性を確認したいと考えている。

さらに、今後の課題として戦略知識の学習がある。

ここでは、ソルバのトレースを追い、不適切な行動を 人間に指摘させることにより、動的に知識を獲得した。しかし、現実には 未熟練者も問題を繰り返し解くことにより順次、能力を増していく。そこに エキスパートがいなくても、経験から学習することができる[Carbonell, et al. 1987] [Mitchell, et al. 1983]。それ故に、システムも不適切な行動を指摘するエキスパートなしで自動的に学習できる可能性がある。

例えば、ソルバが 多くのゴールをスタックしたにも係らず 解が得られない場合、一時的に スタックのゴールをクリアし、その時の状態を初期状態とみなし、問題を解く。そのとき、問題が解ければ、ゴールスタックをクリアできる状態であったと学習する。他方、ゴール・インタラクションがある場合、ＭＥＡの意味で候補でないオペレータを強制的に適用し、ゴールの追求を一時的に遅らせる。このとき、そのオペレータ適用により より良い解を得れることができれば、システムは フォーストール・オペレータを適用すべきケースとして学習する。このような学習方式は、今後の研究課題である。

第７章
リレーショナルデータベースの応用による知識ベースの構築

７．１　まえがき

本章では、表形式で表現される知識を取り上げ、リレーショナルデータベース（ＤＢ）の応用による知識ベースの構築について考察する。オフィスワークを状態遷移モデルで表現したエキスパートシステムＥＬＩＳＥ(Electronic Intelligent Secretary System) を例に、記述の簡易化と記述力の強化の均衡をとるための方式について論じる。

我々の周辺のオフィスワークについて考察すると、大半のオフィスワークは「伝票、帳票などのオブジェクトを白紙のような初期状態から承認済みのような最終状態に遷移すること」とみなせる [Shu. 1985]。これは潜在的に「自動化可能なオフィスワークは状態遷移モデルで表現できる」ことを暗示している。ＥＬＩＳＥは、この状態遷移モデルで表現される小規模、半構造的なオフィスワークの手続きを予め登録しておき、事象（イベント）起動的に実行するものである。ここでいうイベントとは、例えば、時間の到来、メイルの受信などである。

ＥＬＩＳＥは、オフィスワーカの仕事の一部を代行させる秘書システムとみなすことができ、ここではエキスパート＝エンドユーザである。この種のＯＡシステムについては、従来、ＯＢＥ(Office By Example)[Zloof. 1982] のように記述を簡易にすると登録できる手続きが単純となり、逆に SCOOP[Zisman. 1977] のように記述力を強化するとオフィスワーカが自らによる知識を登録・変更が困難となるという問題があった。

本章では、表操作を行うＯＡソフトウェアを使用可能なオフィスワーカを想定し、処理対象に状態に関する管理情報を持たせ、イベントと状態と処理の組をリレーションで表現することにより、表操作言語のカバーする範囲で記述力を確保できることを示す。さらに、ＤＢビュー [Date. 1981][Ullman. 80] を知識ベースビューとして応用することにより登録した知識へのアクセスを簡便化できることを述べる。提案手法の特徴は、表操作という限定処理でオフィスワークの手続きを利用者主導で実行するか、システムに自動的にイベント起動で実行させるかを、適宜入れ替えることが可能なことである。

以下では、はじめに 本章に関連する従来技術と対象とするオフィスワークのモデルについて論じる。次にオフィスの手続きが ４つの関係（手続き関係、イベント関係、アクティビィティ関係、状態関係）で表現できることを示す。その後、ＥＬＩＳＥの６つのコンポーネント（イベントモニタ、手続きモニタ、ディスパッチャ、状態マネージャ、オブザーバ、手続きマネージャ)の機能と役割を それらの相互関係と共に論じる。最後に簡単な予算集計の業務への適用について述べる。

７．２　状態遷移モデルによるオフィスワークの表現

７．２．１　オフィスワーク自動化に関する従来技術

コンピュータとその周辺技術の進歩にも係らず、オフィスワーカの生産性は 製造分野のそれと比較して 改善率が微小であると言われている。ＯＡという言葉が 最近高い関心をもって言及されるのは、このような問題意識を背景に、オフィスワーカの生産性と彼らの作業環境の質を改善することを目的としている。

これまでに ワードプロセッサ、スプレッドシート、ＤＢ検索用の簡易言語など多くのＯＡ用のソフトウェアが開発されてきた。これらのソフトウェアはオフィスの生産性の改善に有効である。しかし、それらの機能は本質的にオフィスで実施されている仕事の機械化であって、手続きの自動化には到っていない。先のソフトウェアを使用するための

（１）いつ、いかなる条件のもとで使用するのか

（２）いかなるソフトウェアをいかなる順番で使用するのか

などはオフィスワーカに完全に依存していた。すなわち、この種の制御情報は個人個人の頭の中の記憶にあるか あるいはマニュアルに記述されているかであり、コンピュータが直接解釈できる形式にはなっていない。

この問題提起は [Zisman. 1978] によりなされている。Zisman は「オフィスオートメーションはタスクの機械化から手続きの自動化へ進むであろう」と予測している。図７．１は このタスクの機械化と手続きの自動化の違いを示している。

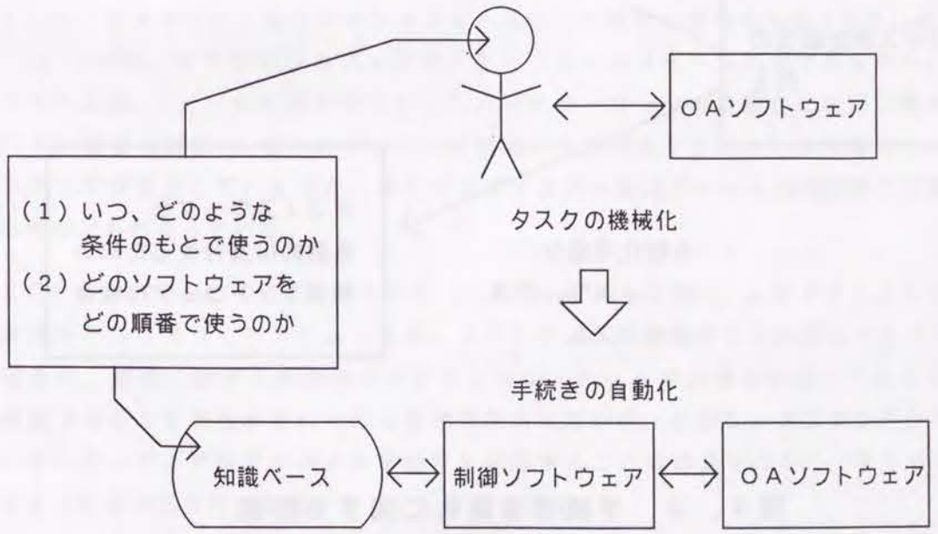


図７．１　タスクの機械化と手続きの自動化

近年、この手続きの自動化に関して多くの研究がなされている。この種の研究は図７．２に示すように次の２つに分類される：

a) オフィスのモデリングの方法論に関する研究

この分野の研究は オフィスの記述、モデリング、分析手法に関するものである。[Ellis, et al. 1980] はＩＣＮ(Information Control Net)と呼ぶ記法を、[Hammer, et al. 1980] はＯＳＬ(Office Specification Language)と呼ぶ記法を提案している。ＩＣＮは 数学モデルに基づき、オフィスの手続きのグラフィカルな表現法を与えている。その結果 オフィスの手続きに潜む制御構造を理解することを可能とし、ボトルネックの検出も可能としている。一方、ＯＳＬはオフィスの機能の分析に基づいて業務の焦点（中心）となるオブジェクトの状態遷移モデルを与えている。ここでオブジェクトは、書類などの具体的なものばかりでなく手続きのような抽象的なものも含む。

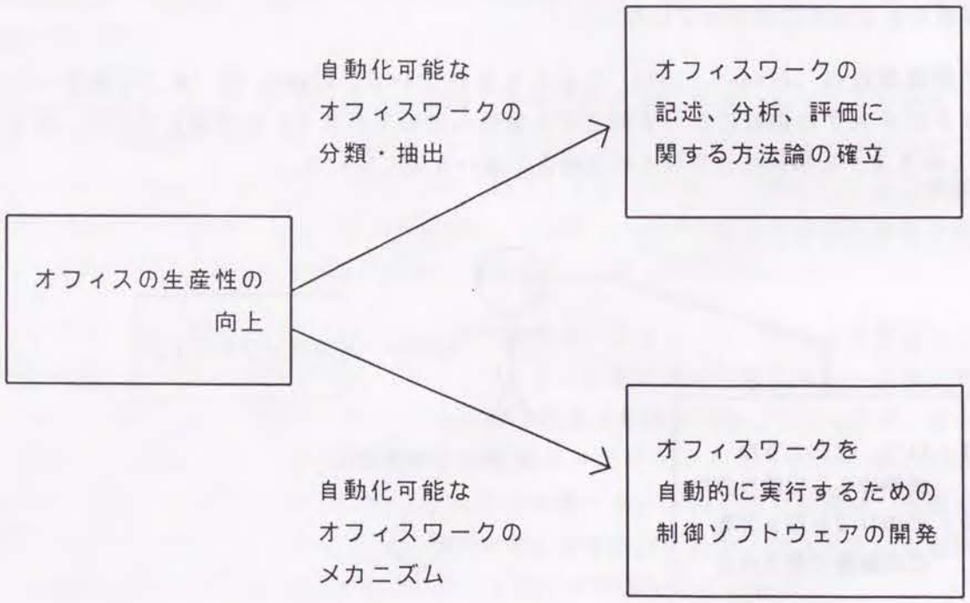


図７．２　手続き自動化に関する研究

b) オフィスの手続きの自動化を実現するシステムに関する研究

これらは、オフィスの手続きの一部を事象起動的に自動実行することに関する研究である。ＯＢＥ(Office procedure By Example) [Zloof. 1982]、ＦＯＲＭＡＬ(Form ORiented MAnipulation Language) [Shu. 1985]、ＯＦＳ(Office Form System) [Tsichritzis. 1982]、ＳＣＯＯＰ(System for COmputerization of Office Procedures) [Zisman. 1977]などが知られている。これらのシステムはデータ処理に基づく手続きの自動化を取り扱っている。

本論は、後者のカテゴリに属すものである。

ＯＢＥ [Zloof. 1982] は、関係データベースのビジュアル・インタフェースであるＱＢＥ (Query By Example) の拡張である。ＯＢＥは データベースだけでなくワードプロセッシング、電子メイル、トリッガ・プログラムなどのアプリケーションを統合したものであり、ノンプログラミング・ユーザにとって利用しやすいものである。

ＯＢＥは暗に、「処理されるオブジェクトは 常に一定の状態にある」ことを仮定している。しかし、一般に オブジェクトは 状態を変化しながら処理されるべきである。例えば、ある書類は承認されているか、承認されていないか、また、ある条件のもとで承認されているか、さらには上司からコメントを付加されて返却された状態であるか などを考慮する必要がある。それ故に、オブジェクトの状態を区別していないＯＢＥのオフィス手続きの自動化機能は、かなり制限の強いものとなっている。

ＦＯＲＭＡＬ[Shu. 1985] およびＯＦＳ [Tsichritzis. 1982] はフォーム（定型用紙）に基づくシステムである。フォームは オフィスの中で中心となるオブジェクトであると考えられ、システムと人間とのインタフェースとして極めて適当なものである[Lefkovitz, et al. 1979]。大半のビジネス・アプリケーションはフォーム処理であるから、これらのシステムは フォーム処理を中心としたルーチン・ワークの自動化に検討の焦点を当てている。重要な課題は 種々のフォームの構造的な表現法である。しかしながら、ＯＡソフトウェアが普及している現在、多くのオフィスワーカはフォーム処理以外に文書処理、表処理なども行なっている。

ＳＣＯＯＰ[Zisman. 1977] はＡＰＮ (Augmented Petri Net) と呼ぶオフィスの手続きの表現モデルに基づくシステムである。ＡＰＮでは、処理に関する知識はペトリネットで表現され、制御に関する知識はプロダクション・ルールで表現される。これらの表現は状態遷移モデルを表現するに十分な記述力を与えている。しかし、ノンプログラミング・ユーザにとってＡＰＮでオフィス手続きを記述することは容易ではなく、またＳＣＯＯＰに与える記述の正当性を検証することも容易ではない。

さらに、従来のこれらのシステムは自動化される手続きとオフィスワーカが主導で起動する手続きの入替えが容易ではなかった。

ＥＬＩＳＥの開発目標は以下のとおりである。：

（１）ＯＢＥより広いクラスのオフィスワークを自動化の対象とする。
（２）対象とするオフィス手続きとしてＯＡソフトウェア全般に渡るように広いクラスのものを扱う。
（３）ノンプログラミングユーザが手続きを指定することができ自動化するオフィス手続きの検証を行なう機能を提供する。
（４）手続きに関して、システムにイベント起動で実行させるか、オフィスワーカ主導で実行するか を適宜入れ替えることを可能とする

７．２．２　自動化対象のオフィス手続きモデル

オフィスワーカは ＥＬＩＳＥに オフィスワークの自動化可能な部分についていかなる処理を行うかを教示しなければならない。ここで、オフィスワーカは ノンプログラミング・ユーザであると考える。従って、従来のプログラミング言語以外の記法を採用しなければならない。すなわち、オフィス手続きの自動化可能な部分について処理方法に関する知識の簡便な表現法が必要となる。

オフィス手続きは、「あるオブジェクトを初期状態から いくつかの中間的な状態を経由して 最終状態に遷移させる一貫した処理」とみなすことができる。例えば、旅費精算の処理においては、出金伝票は 始めの白紙の状態から 最終の出金完了の状態までに、

記入待ち、上長の承認待ち、内容の確認待ち、元簿への転記済み

などの状態を経由する。それゆえに、状態遷移表現は オフィス手続きの自動化の基礎となる。

オフィス手続きは 図７．３に示すようなオートマトンとみなすことができ、それをルールで表現すると次のようになる:

IF event WHEN state-1 THEN process GOTO state-2.

これは、「もし "event" が起こったならば、"state-1" のオブジェクトに対し、 "process" を施し、その状態を "state-2"に変える」ということを表わしている。

次に本章における用語の定義を行なう。

定義１：オブジェクト

一つのオブジェクトは オフィスワークを進行する上での処理対象である。処理対象は それが処理されたとき その状態を変更する。フォーム、シート、ファイルなどが具体的なオブジェクトである。

定義２：イベント

イベントは 処理を起動する要因である。あらかじめ指定しておいた日時の到来、メイルの受信、ユーザによるコマンド入力、指定したカウンタの更新、オブジェクトの状態遷移などがイベントの例である。

定義３：状態

状態はオブジェクトに付加される管理情報である。最終状態を除いて他の状態にあるオブジェクトはイベントの発生を待っていると考える。イベントが発生したとき、オブジェクトは 状態に依存してあらかじめ定義された処理を施される。

定義４：処理

処理とはオブジェクトに対する操作列である。一つの処理は一連のアクティビィティからなる。一つのアクティビィティはＯＡソフトウェアの一つのコマンドに対応し、データベース検索、表処理、文書処理、メイル処理などを行なう。

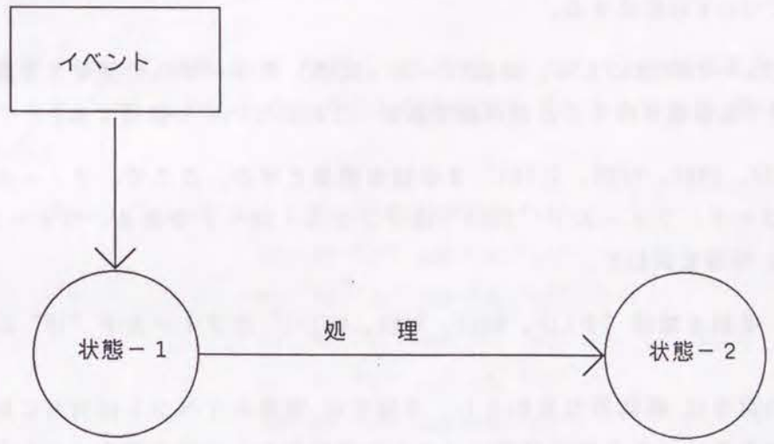


図７．３　オートマトンによるオフィスワークの記述

7．3　関係による知識の表現と知識ベースビュー

7．3．1　オフィス手続きの表現と蓄積

オフィスワークの状態遷移表現を 関係データベースに蓄積する。ここで、関係データベースを利用するのは 次の４つの理由による。

（1）状態遷移モデル（IF event WHEN state-1 THEN process GOTO state-s）は直接、関係P(IF, WHEN, THEN, GOTO) で表現できる。ここで、関係の定義は テーブルへの値入力であり プログラムの記述よりも容易である。

（2）関係代数の使用により、種々の側面から登録した手続きを検索することが可能である。例えば、ある状態に対し、いかなる処理が施されるかを検索したり、ある日時に行われる可能性のある処理を検索することなどが、データベース検索コマンドを使用することにより可能である。

（3）知識の不足・重複などの整合性をとる検証がデータベースコマンドの使用により可能である。この種の機能は、知識ベースを継続的に保守していく上で重要である。この例については後述する。

（4）ビューの利用により、知識ベース（関係）のユーザへの見せ方を設定したり、更新に対する保護を行うことが可能である。これについても後述する。

関係 "P(IF, WHEN, THEN, GOTO)" を手続き関係と呼ぶ。ここで、フィールド "IF" はイベント・コード、フィールド "THEN" はプロセス・コードである。フィールド "WHEN" と "GOTO" は 状態を表わす。

はじめに、手続き関係 "P(IF, WHEN, THEN, GOTO)" のフィールド "IF" について考察する。

イベントの発生は 瞬間的なものとし、本論では 複数のイベントは同時に起こらないと考える。すなわち、「ある指定日時にメイルを受け取る」という複合イベントは、「その時刻の到来」と「メイルの受信」という二つのイベントの組み合わせと考える。つまり、適当な状態を導入することにより、複合イベントの記述を代替し、 それに対応する手続きを記述する。

イベントの発生は、イベントの種別、値とその論理関係の組で監視される。例えば、「あるイベントは "日時" が "２月２１日" "以降" である」などである。これより、関係 "P" のフィールド "IF" を関係 "E(Eid, Type, R, C)" として詳細定義する。ここで、 "Eid" はイベント・コードであり、その値域は 関係 "P" のフィールド "IF" と同一である。フィールド "Type" はイベントの種別（この例を表７．１に示す）である。

表７．１　イベントの種類

タイプ	説明
コマンドイベント	ユーザにより特定のコマンドが使用されたとき
受信イベント	電子メイルを受け取ったとき
時間イベント	ある日付け、ある時刻の到来
状態イベント	処理対象がある状態に遷移させられたとき

関係 "E(Eid, Type, R, C)"（以後、イベント関係と呼ぶ）のフィールド "R" は論理演算子であり、フィールド "C" は値を表わす。ここで、論理演算子 "R" は次のいづれかである：

等しい（＝）、　より大きい（＞）、　以上である（≧）
等しくない（≠）、　未満である（＜）、　以下である（≦）

イベント種別 "type1" が値 "c1" で起こったとき、このイベント関係を用いて 手続き関係のフィールド "IF" に対応するイベントが定義されているか否かが 次の検索式を用いて決定される：

```
Type= "type1" and ( R= "＝" and C＝ "c1"
                 or  R= "＞" and C＞ "c1"
                 or  R= "≧" and C≧ "c1"
                 or  R= "＜" and C＜ "c1"
                 or  R= "≦" and C≦ "c1"
                 or  R= "≠" and C≠ "c1")
```

次に手続き関係 "P(IF, WHEN, THEN, GOTO)" のフィールド "THEN" について 考察する。

処理の分岐は イベントの判別で行なうので、実行に関しては図７．４に示すような直列と並列の処理を考えればよい。この両者を表現するため、関係 "A(Pid, Pp, Ps, Aid)" を定義する。ここで、フィールド "Pid" の値域は、関係 "P" のフィールド "THEN" と同一である。フィールド "Pp" は 並列処理の識別を表わし、フィールド "Ps" は直列処理の識別を表わす。フィールド "Aid" にはＯＡソフトウェアのコマンドが定義される。この関係をアクティビィティ関係と呼ぶ。この関係" A" は ある種のマクロコマンドを定義しているとみなすことも可能である。

最後に オブジェクトの状態管理について 考察する。各オブジェクトには 状態が付加される。そして イベントの発生の都度、その状態と定義されている知識に依存して、次の状態へと遷移する。関係 "S(Oid, Sid)" は、フィールド "Oid" でオブジェクトのコード、フィールド "Sid" で状態を関連つけて管理するものである。これを状態関係と呼ぶ。

以上により、オフィス手続きは、4種類の関係"P(IF, WHEN, THEN, GOTO)"、
"E(Eid, Type, R, C)" "A(Pid, Pp, Ps, Aid)" "S(Oid, Sid)" で表現され、リレーショナル・データベースに蓄積できることが示せた。

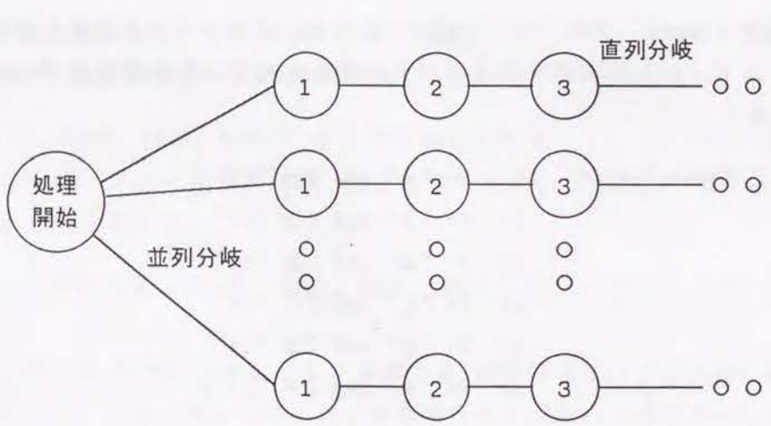


図7．4　オフィスワークにおける並列処理と直列処理

7．3．2　コンポーネントの機能と役割

次に、定義した4つの関係を参照して動作する ＥＬＩＳＥの機能コンポーネントについて述べる。

あるオブジェクトを初期状態から最終状態に遷移させる一連のアクティビィティの中で、ある部分は自動化され、残りの部分はオフィスワーカにより起動されると考える。後者は複雑な意思決定や例外処理を含んでいる。半構造的なオフィスワークの自動化には オフィスワーカとシステムの協調が必要である。システムは自動化可能な部分のみ、その仕事を代行することにより オフィスワーカを支援する。また、自動化される部分とオフィスワーカに起動される部分について簡便に変更される必要がある。

これらの要求を考慮すると、ＥＬＩＳＥとワードプロセッシングや電子メイルなどのＯＡソフトウェアの関係は図7．5のようになる。図のようにＥＬＩＳＥは大きくオフィス手続きを獲得する部分と実行する部分からなる。また、ＯＡソフトウェア群にたいしては、オフィスワーカへの直接のインタフェースとＥＬＩＳＥが起動するインタフェースがある。

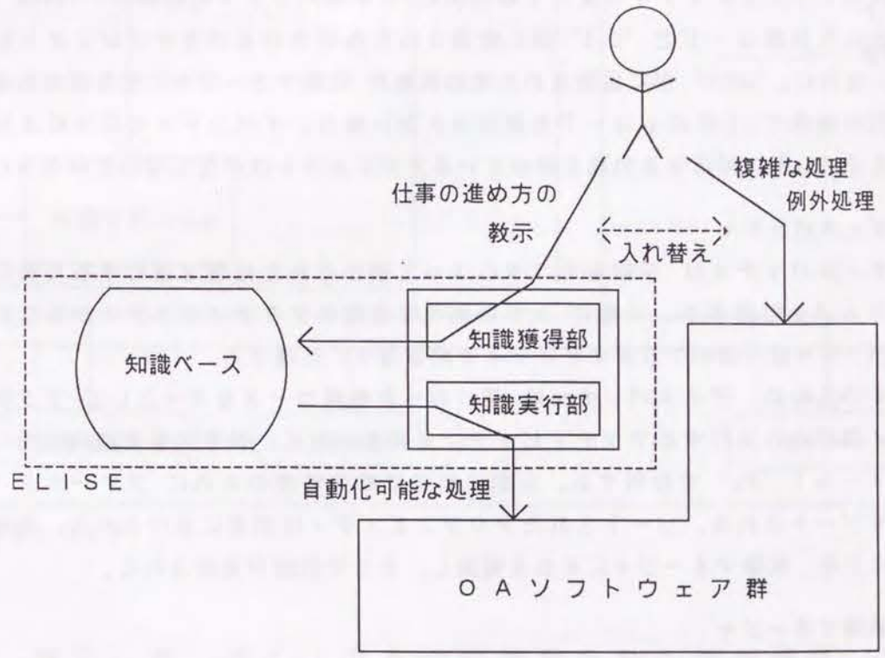


図7．5　ＥＬＩＳＥとＯＡソフトウェアの関係

イベントの発生により事象起動的に処理するために、ＥＬＩＳＥの知識実行部のコンポーネントは 図７．６のようになる。各コンポーネントの役割は 次のとおりである。

（１）イベント・モニタ

イベント・モニタは 指定された日時の到来、メイルの受信、利用者によるコマンド投入などのイベントの発生を監視する。イベントが発生したとき、イベント・モニタはそれが手続きを起動するものであるか否かをイベント関係を検索することにより決定する。このときの検索条件は ７．３．１で述べたとおりである。検出したイベントが、オフィス手続きにトリッガをかけるものであれば、それを次の手続きモニタに引き渡す。

（２）手続きモニタ

本コンポーネントは 受理したイベントにより、起動される手続きと処理対象オブジェクトを検索する。このために 手続きモニタは 手続き関係と状態関係を "WHEN" および "Sid" をキーとして ＪＯＩＮ操作 [Date. 1981] を行ない、状態マネージャに問合せることにより、処理される状態にあるオブジェクトのリストを得る。

処理されるべきオブジェクトが一つ以上存在すれば、手続き関係 "Ｐ" に定義されているアクティビィティを実行するために ディスパッチャを起動し、"THEN" 部に定義された処理コードと "Oid" 部に定義された処理されるべきオブジェクトを引き渡す。さらに、"GOTO" 部に定義された次の状態が 状態マネージャに引き渡される。

先の検索で １件のレコードも検索されない場合、イベント・モニタにより検知されたイベントに対応する処理を待っているオブジェクトは存在しないとみなされる。

（３）ディスパッチャ

ディスパッチャは 手続きモニタによって検出された処理を遂行するためＯＡソフトウェアを起動する。一般に 一つの処理は複数のアクティビィティからなる。ディスパッチャは一連のアクティビィティを順番通りに処理する。

そのために、ディスパッチャは 受け取った処理コードをキーとして アクティビィティ関係から実行するアクティビィティを得る。次に、検索結果を並列処理のためにフィールド "Pp" で分割する。分割の結果は直列処理のために フィールド "Ps" によりソートされる。ソートされたアクティビィティは順番に実行される。処理が完了したとき、状態マネージャにそれを報知し、そこで状態が更新される。

（４）状態マネージャ

本コンポーネントは オブジェクトの状態を管理する。あるイベントが発生し、対応する処理がそのオブジェクトに施されたとき、状態マネージャは手続きマネージャから引き渡された情報をもとにオブジェクトの状態を更新する。

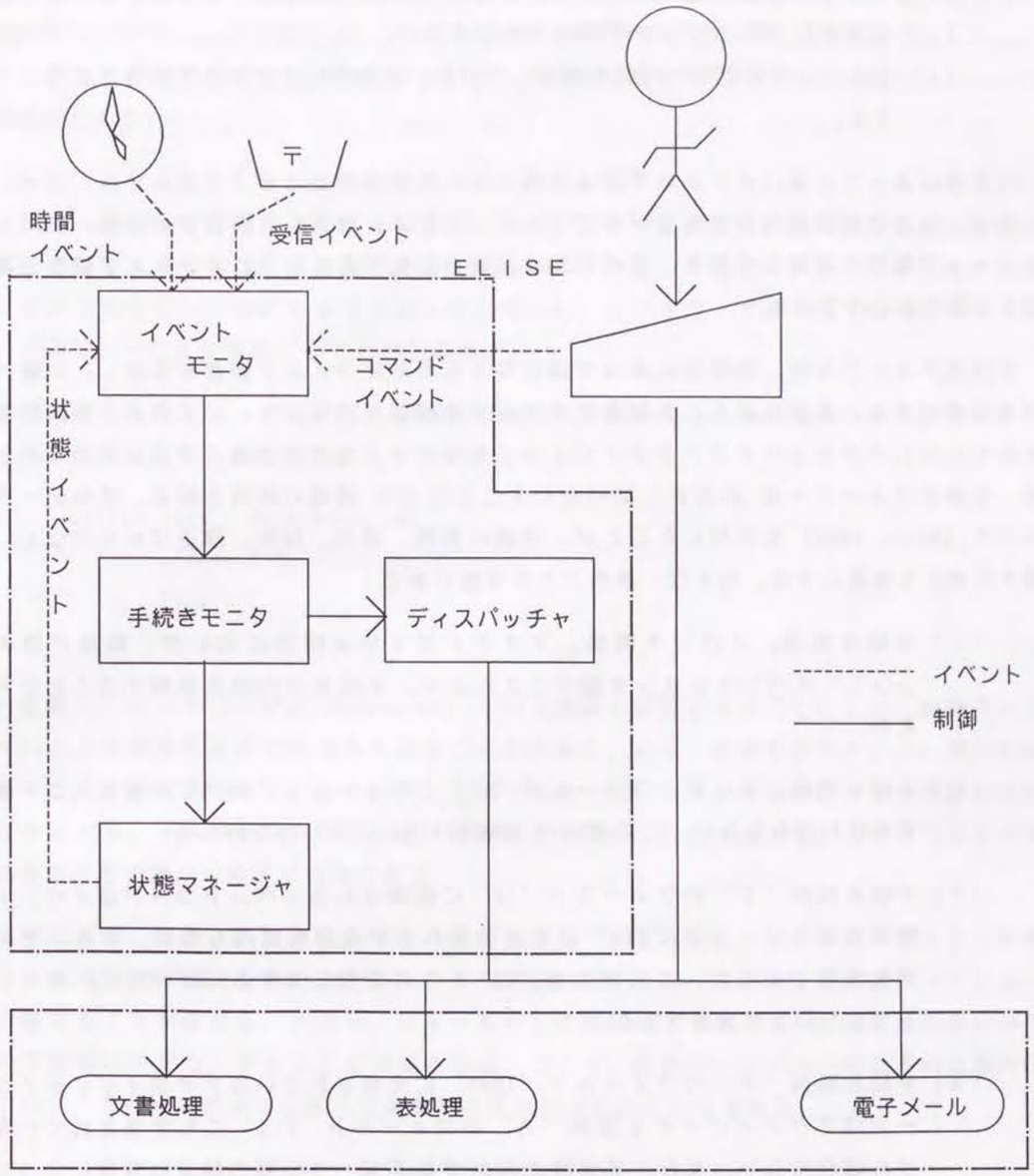


図７．６　ＥＬＩＳＥの知識実行部の機能構成

一方、ＥＬＩＳＥの知識獲得部については さらに 次の２つの部分からなる。

（ａ）利用者が 陽にオフィス手続きを教示する。

（ｂ）システムが利用者の行動を観察しておき、自動的にオフィス手続きを獲得する。

利用者によって、陽にオフィス手続きを教示される部分のことを「手続きマネージャ」と呼び、後者の自動獲得の部分を「オブザーバ」と呼ぶ。後者は、利用者の行動、使用したコマンド履歴を観察しておき、その行動を以後自動化できるようにオフィス手続きの蓄積を支援するものである。

手続きマネージャは、利用者によって陽に与えられたオフィス手続きを受諾し、知識ベースに記憶する。必要に応じ、手続きマネージャは編集用のスクリーンを表示する。定義されていないオブジェクトやアクティビィティを検出するなどの矛盾、不足を検出したとき、手続きマネージャは 利用者に問い合わすことにより 問題の解消を図る。関係データベース [Date. 1981] を利用したことが、情報の蓄積、追加、削除、修正ばかりでなく、種々の検証を容易にする。例えば、次のことが可能である：

（１）手続き関係、イベント関係、アクティビィティ関係に対して、関係代数のＪＯＩＮオペレーションを施すことにより、手続きの内容を参照することができる。

（２）手続き関係において、フィールド "IF" とフィールド "WHEN" の組はユニークでなければならない。この検証は PROJECTION により行なわれる。

（３）手続き関係 "P" のフィールド "IF" に定義されるイベントコードはイベント関係Ｅのフィールド "Eid" にも定義されていなければならない。また、その逆も同様であるが、この種の検証は ２つの関係に対する DIFFERENCE オペレーションにより実施できる。

（４）手続き関係 "P" のフィールド "THEN" に定義されているアクティビィティコードはアクティビィティ関係 "A" のフィールド "Pid" にも定義されていなければならない。また、その逆も同様であるが、この種の検証は同様に２つの関係に対し DIFFERENCE オペレーションをとることにより実施できる。

７．３．３　ビューによる役割分担の変更

ＥＬＩＳＥが対象とする小規模、半構造的なオフィスワークの手続きは一般に頻繁にその内容が変更される。一度、機械化した業務にたいし、例外的な環境変化があり、人間が処理に介在しなければならない場合もある。一方、当初は自動化不可能と考えていた例外的な業務が定着し、あるときからシステムに処理を委託したいという場合もある。

従来、このような変更要求にシステムが対応困難であったため、オフィスの自動化が製造分野の自動化に比べ進まなかったと考える。すなわち、第３章でも述べたようにＥＤＰ部門のプログラミングに対して、後でエンドユーザ部門が変更することが困難であったし、エンドユーザ部門のプログラミングは、ユーザが常に起動をかけなければならないという問題が合った。

それに対し、ＥＬＩＳＥは知識をリレーショナルデータベースに蓄積しているので、データベースビューを利用することにより、エンドユーザに変更される可能性のある部分のみにアクセスさせることが可能である。すなわち、一度、自動化されるオフィスワークがナレッジエンジニアなどにより定義された場合に、エンドユーザが変更する対象をデータベースビューにより規定することが出来る。

例えば、エンドユーザにより、継続的に知識の変更がなされるが、その変更対象が日付イベントに限定される場合には

```
define view   date-event as
       select Eid, Type, R, C
       from   E
       where  Type=date
```

と定義し、エンドユーザに date-event という関係を参照させることにより、知識ベースのいかなる部分を変更できるかを示すことが出来る。逆に、変更が許可されない部分を保護することが可能である。さらにデータベースは一般にカラムごとにアクセス権限が設定されるので、イベントコードの追加・削除を許す場合、日付の値（Ｃ）のみの変更を許す場合などきめ細かい指定が可能である。

すなわち、関係という表現方法を一種の知識モデルとみなし、エンドユーザからみたオフィスワークの構造を専門家モデルとみなすと、データベースビューは知識ベースビューと考えることが出来る。つまり、ビューを介した操作により利用者には抽象度の高いレベルで知識にアクセスすることが可能となる。ここで、知識ベースビューの形式的定義は以下のデータベースビューの形式定義をそのまま利用することができる。

```
<ﾃﾞｰﾀﾍﾞｰｽﾋﾞｭｰ定義> === define view <ﾋﾞｭｰ名> as
                                <SQL-select文>
<SQL-select文> === select <ｶﾗﾑ名>{,}...
                  from   <ﾃｰﾌﾞﾙ名>{,}...
                 [where  <検索条件節>{,}...]
```

（注）ＳＱＬ－ｓｅｌｅｃｔ文については [Date, 1981]が詳しい。

7．4　予算集計業務への適用例

リレーショナルデータベースの応用により知識ベースを構築する利点を示すために、ある部長の秘書の仕事を例にとって 提案したオフィスワークの自動化方式について考える。この秘書の仕事は 期末に今期の予算消化の見積りを所属している課長に問い合わすものである。

ＥＬＩＳＥにオフィス手続きを定義するために次の４つのステップを踏む：

ＳＴＥＰ １：オフィスワークの記述

（１）期末の１ケ月前（２月２１日），この秘書は各課長に残り１ケ月での支出の見込を問い合わす用紙を送付する。

（２）課長から見込が記入されて返送されたら、それを集計用紙に転記する。

（３）問合せ用紙を送付したのち、期限（例えば、２月２４日）到来後も返答がなければ、返答を行うように督促状を送付する。

（４）全ての課長からの返事が揃えば、集計用紙の上で計算を行ない、部の見積りとして取りまとめる。

ＳＴＥＰ ２：関与するオブジェクトの同定

この手続きでは集計用紙が焦点を当てておくべきオブジェクトである。

ＳＴＥＰ ３：関与するイベントの同定

この手続きでは次の３つのイベントが関与する:

e1: 日付の到来，"２月２１日",

e2: 日付の到来，"２月２４日",

e3: 秘書への返事の到着．

ＳＴＥＰ ４：中心となるオブジェクトの状態の同定

集計用紙の状態:

- ２月２１日以前
- 返事が一通も来ていない状態 (waiting),
- 返事が一通来ている状態 (one-received),
- 返事が二通来ている状態 (two-received),
- ３人の課長全てから返事が来ている状態 (end).

このように考えたときの手続きの例を 表７．２から表７．５に示す。ここでプロトタイプシステムのＯＡソフトウェアには 日立の VOS3/EXCEED (Exective Management Decision Support System)[Isobe, et al. 1982]を採用している。ＥＸＣＥＥＤは、データベース操作、統計分析、ビジネスグラフ、メイリングなどの機能を有すエンドユーザ向けの簡易言語である。但し、表７．４に示したコマンドのシンタックスは読みやすさを考えて実際の構文から少し変更している。

表７．２　手続き関係の例

ｺｰﾄﾞ	遷移前状態	処理	遷移後状態
e1	not-active	a1	waiting
e2	waiting	a2	waiting
e2	one-received	a2	one-received
e2	two-received	a2	two-received
e3	waiting	a3	one-received
e3	one-received	a3	two-received
e3	two-received	a4	end

表７．３　イベント関係の例

ｺｰﾄﾞ	タイプ	関係	値
e1	日付	=	２月２１日
e2	日付	=	２月２４日
e3	受取	=	メモ-1

表７．４　アクティビティ関係の例

処理	区分	実行順序	処理コマンド
a1	p11	1	create memo-1 for section-a
a1	p11	2	send memo-1 to section-a-manager
a1	p12	1	create memo-1 for section-b
a1	p12	2	send memo-1 to section-b-manager
a1	p13	1	create memo-1 for section-c
a1	p13	2	send memo-1 to section-c manager
a2	pa2	1	send memo-2 to sesction manager who did not return memo-1
a3	pa3	1	load work-sheet from file-0
a3	pa3	2	copy memo-1 to work-sheet
a3	pa3	3	save memo-1 to file-1
a3	pa3	4	save work-sheet to file-0
a4	pa4	1	load work-sheet from file-0
a4	pa4	2	copy memo-1 to work-sheet
a4	pa4	3	calculate work-sheet
a4	pa4	4	save work-sheet to file-0
a4	pa4	5	save memo-1 to file-1

表７．５　状態関係の例

ｵﾌﾞｼﾞｪｸﾄ	状態
集計表	not-active

この例から、以下の利点を確かめることができる。

（１）まず手続き関係を定義し、その後 他の３種の関係を定義することにより、トップダウンに知識を記述することが可能である。特に、処理 a2，a3 は複数の手続きで利用されているがそれは別途マクロ・コマンドとしてアクティビティ関係に定義される。

（２）処理の制御はすべて関係の中に記述されるので、通常のプログラミング言語で記述するより簡便に知識を記述できる。

（３）データベース管理システムが具備する検索コマンドを活用することにより、種々の観点から定義されている知識を検索することが可能である。さらに、手続き関係に定義されたコード（e_i，a_i など）が他の関係で具体化されているか否かの検証もＤＢの検索コマンドを利用することにより可能である。

（４）例外的に２月２３日に督促を行なう必要がある場合、利用者が直接ＯＡソフトウェアのコマンドを発行することが可能である。

（５）このような業務が繰返し実施される場合、ビューを設定し、イベント関係の日付のみを変更可能とし、エンドユーザに提供することができる。

7．5　まとめ

本章では、表形式で知識表現されるＯＡ用のエキスパートシステムの構築に関して論じた。このエキスパートシステムは、小規模、半構造的なオフィスワークを部分的に自動化することを目標とするものである。自己の仕事の一部を代行させる秘書システムとみなすことができ、エキスパート＝エンドユーザである。

本章の成果は 表操作言語のカバーする範囲で記述力を確保し、データベースビューを知識ベースビューとして応用することにより 登録した知識へのアクセスを簡便化したことにある。ここでの特徴は、表操作だけでオフィスワークの手続きを利用者主導で実行するか、システムにイベント起動で実行させるかを適宜入れ替えるなどの保守が可能なことである。

まず、ＯＡの推進には ワードプロセッサやスプレッドシートで実現した仕事の機械化だけでなく 処理の自動化が必要であるが、従来の処理の自動化に関する技術では 記述力と記述の容易性から限界があることを述べた。次に オフィスワークを状態遷移モデルで捉えると知識を関係として表現できることを示した。この表現は、プログラム非専門家にとって、記述が容易なものであり、関係表現は蓄積した手続きの検索、追加、編集、削除だけでなく、検証にも都合のよいものであることを示した。

さらに、関係として定義した知識に対して、データベースビューを設定することにより、ユーザがアクセスする知識の範囲を限定できることを示した。限定された範囲で知識ベースの保守を行ったり、試行を行うことができる。また、本論で設計した知識の関係構造に拡張が必要な場合も データベースシステムが具備する機能を利用することにより変更可能であり、拡張した後もデータベースビューの設定により 知識のうち 変更されない部分の保護が可能である。すなわち、データベースビューをそのまま知識ベースビューとして利用することが可能である。

続いて、オフィスの手続きを自動化するためのアーキテクチュアについて論じた。これはオフィスの手続きを関係として表現し、データベースに蓄積しておき、事象起動的に実行するものである。

提案したアーキテクチュアに基づいてＥＬＩＳＥと呼ぶプロトタイプを開発した。これは、イベント・モニタ、手続きモニタ、ディスパッチャ、状態マネージャ、手続きマネージャ、オブザーバからなる。各コンポーネントの機能とコンポーネントの間のインタフェースについて詳細に論じた。

最後に、記述した機能について、簡単な予算集計業務を例にとり示した。提案した方式では、オフィス手続きを獲得、蓄積、実行する上で、システムと利用者が協力して遂行できる点に特徴がある。

第８章　結　　論

本論文では、エキスパートシステム構築の関与者と構築手順を分析し、知識ベースビューの概念を導入することにより、エキスパートシステムの開発、再利用のモデルを提案し、具体的な応用例に基づいて、知識ベースビューの表現、機能、有効性、限界について考察した。

知識ベースビュー導入の目的は

① 応用領域に関して汎用的な知識表現とエキスパートのもつ問題解決に関する知識表現の抽象レベルの変換を行う、

②エキスパートシステム開発時に、不正確かつ不完全であった知識をエキスパート自らが洗練化していくことを可能とする、

ことにある。

本論では 特に、知識が不完全な時からエキスパートシステムが運用されることを前提とした問題解決モデルについて考察し、誰がどうエキスパートシステムの構築に参加し、同一の知識ベースに対して、誰がどう見るかなどの運用面の課題について研究した。その結果として、具体的に、以下の結論を得た。

第３章では、計算機部門によるプログラミングとエンドユーザ部門によるプログラミングとを対比することにより、エキスパートシステムの構築環境について考察した。環境として、試行環境、開発環境、利用環境、保守環境が必要であることを導き、それらを互いに関連付けるために、知識ベースビューが有効であることを示した。知識ベースビューは開発環境の上に他の３つの環境を統合するためのものである。

知識ベースビューの導入により、ナレッジエンジニアが開発した知識ベースをエキスパートが保守することが可能となり、さらに、一度構築されたエキスパートシステムのアーキテクチュアが類似のエキスパートシステムを試行することが可能となる基本的な構想が確立された。

第４章では、第３章の構想に基づき、知識ベースを静的に保守するための知識ベースビューについて論じた。

知識ベースの構造と属性に２つの仮定をおき、それを基にナレッジエンジニア用の知識表現とエキスパート用の知識表現を相互に変換する方式を述べた。

本方式に基づき生成される知識ベースエディタは、知識ベースビューとしてエキスパートが知識ベースのどの部分にアクセスできるかを示し、フィルインザブランク形式でエキスパートの操作を誘導し、重複した知識や冗長な知識の警告を行う。この知識ベースビューは、新たな問題解決モデルが考案される都度生じていた個別に知識獲得ツールを開発する労力を省力化した。

第５章では、第４章で提案した知識ベースビューの応用例として、プログラミング・ノウハウを伝承するためのエキスパートシステムの構築について論じた。

はじめに本エキスパートシステムが必要とされるソフトウェア開発の現状について述べ、問題を解決する専門家モデルを提案した。このモデルを計算機上に実現するための知識表現について述べ、収集した知識を「継続的に保守される部分」と「基本的に変更されない部分」に分けた。継続的に保守される部分について構造と属性を定義し、第４章で提案した知識ベースビューで エキスパート自らが知識を保守することを可能とした。この種のアプリケーションでは 完全なルールを短期間で得ることが困難であるが、エキスパートが気付いたときに新たなルールを入れることが重要と考えた。

さらに、知識ベースビューが エキスパートシステムのアーキテクチュアをリサイクリックに利用可能とすることについても示した。

第６章では、第３章の構想に基づき、知識ベースを動的に保守するための知識ベースビューについて論じた。まず、はじめに従来知られていた戦略知識では、質の良い解を効率良く求められないことを示し、新たな２種類の戦略知識を示した。

続いて、これらの新しい戦略知識を問題解決時に獲得する方式について論じた。この方式は、システムの動作に誤りがあると気付いたときに、そのときの状態を一般化することにより戦略知識を獲得するものである。アルゴリズム的には種々の知識を同時に獲得できるが、与えられた問題の複雑さによって

（１）ある側面の戦略知識に焦点を当てたほうが獲得しやすいこと、

（２）そのとき、知識ベースビューという概念を導入することが有効であること、

を論じた。

この方式をエイトパズルの問題に適用し、実際にシミュレーションすることにより、従来の方式に比べ、探索ノードを２３％削減し、解の長さを２７％短くなることを確認した。戦略知識については 新たな種類のものが 今後 発見され ますます多様になる可能性があるが、このとき知識ベースビューが、ある一面の戦略知識に焦点を当てて獲得することを可能にする。

第７章では、表形式のデータを操作するＯＡソフトウェアをベースとしたエキスパートシステムの知識表現と知識ベースビューについて論じた。

まず、ＯＡの推進には ワードプロセッサやスプレッドシートで実現した仕事の機械化だけでなく 処理の自動化が必要であるが、従来の処理の自動化に関する技術では 記述力と記述の容易性から限界があることを述べた。次に オフィスワークを状態遷移モデルで捉えると知識を関係として表現できることを示した。

状態遷移モデルを導入することにより記述力を確保し、関係表現することによりＤＢビューを知識ベースビューとして応用した。その結果、登録した知識へのアクセスが簡易となり、表操作だけでオフィスワークの手続きを利用者主導で実行するか、システムにイベント起動で実行させるかを入れ替えることなどが可能となった。

エキスパートシステムの構築に関しては まだ多くの研究の余地が残っている。提案した知識ベースビューに関連するいくつかの項目を述べると次のようになる。

（１）知識獲得には、知識の入力だけでなく、その正確性、十分性を保証するための検証技術が必要である。

重複した知識、矛盾した知識、冗長な知識の検出は、その一部を本研究で触れたが、まだ、十分ではない。さらに 不足した知識を見つけ、それを積極的にエキスパートに求める方式についても 今後 研究に値すると考えている。

（２）システムが自動的に知識を獲得する機械学習の研究がより期待されるであろう。

本論で述べた知識獲得は、基本的にエキスパートが知識をいれることを前提としている。しかし、問題解決の経験と共に 自動的に知識を洗練していく能力をシステムにもたせることは魅力的な研究テーマである。例えば、４章で述べた方式については、知識ベースの構造を認識して、ＦＯＲＭを自動的に生成する試みが考えられる。第６章については、そのまとめでも言及したが、戦略知識を獲得した状態の一般化を自動的に行うことができれば、学習といえるであろう。さらに第７章に関していえば、利用者のＯＡソフトウェアの使用履歴から なんらかの業務知識を獲得することができる可能性がある。

（３）知識ベースのある側面だけを与えられて、その全体構造を作成することも重要な課題である。

本論では、複雑な構造をした知識ベースのある側面を知識ベースビューとしてみることのみを考えた。比喩を用いれば、立体図形があるとき、その展開図の見方を論じたようなものである。逆に、展開図だけが与えられたとき、それから立体図形を構築するようなことも考えるべきであろう。すなわち、ビューとしての知識を集めることにより、対象とする知識の全体構造を決定する方式の研究も重要なテーマとなろう。

（４）分散環境での知識ベースビューを考慮していく必要がある。

本論では、１ヶ所に集中した知識ベースについて考察した。しかし、知識は複数の人に分散していることが一般的であるし、また、システム実現上からネットワークに分散した知識ベースを取り扱う技術が必要とされるであろう。この場合も、ビューという考え方で、物理的な分散と論理的な分散を区別していくことが有効であろうと思われる。

謝　　辞

本論文は、著者が 昭和５８年から現在に到るまでの期間で、㈱日立製作所 システム開発研究所において行った エキスパートシステム構築に関する研究の成果をとりまとめたものである。

京都大学工学部情報工学科 教授 堂下修司先生には、数々のご助言を頂き、本論文をまとめる上で多大なご指導、ご尽力を頂いた。また、同 数理工学科 教授 片山徹先生には、著者が京都大学在学中、卒業研究、修士論文をまとめるに当たり ご指導頂くと共に研究生活の端緒を開いて戴き、卒業後も研究にたえず激励を頂いた。さらに 同 情報工学科 教授 松本吉弘先生、助教授 西田豊明先生には本論文を草する上で貴重なご教示を頂いた。これらの先生方に深くお礼を申し上げます。

また、本論文の第６章は、著者が米国 Carnegie-Mellon 大学に滞在中行ったものである。当時指導頂いた Jaime G. Carbonell 教授、 Steven Minton 博士にはプロブレムソルバＰＲＯＤＩＧＹを使用させて頂くと共に数々の議論をさせて頂いた。合わせて感謝します。

本研究は ㈱日立製作所 システム開発研究所において、所内外の多数の方々のご指導ご助力を得て行われた。特に システム開発研究所 歴代所長、三浦武雄博士、川崎淳博士、堂免信義氏には、海外での研究生活を含め、本研究の機会を与えて頂いた。本論文をまとめるに当たり、三巻達夫博士、石原孝一郎博士、絹川博之博士には、終始 叱咤激励頂くと共に、細部にわたり 丁寧にご指導頂いた。三森定道博士、大成幹彦博士からは、ものの見方、考え方など研究者の基本姿勢を学んだ。

同所 前第５部長 森文彦博士（現ソフトウェア開発本部）には、著者の入社以来の直接の上司としてご指導ご鞭撻頂き、研究生活に常に激励を戴いた。論文をまとめるにあたり 暖かいご支援を戴いた。著者は、同博士から、企業内研究について 立案の仕方、進め方、まとめ方などを学んだ。同博士の指導なしでは、著者は一つの論文も書くことはできなかったであろう。青山義彦氏、春名公一博士、佐藤敬博士にも 著者の上司として 研究一般に関するご指導を頂くと共に良い研究環境を与えて頂いた。

さらに、同所の増位庄一氏、中所武司博士、近藤秀文氏、広瀬正氏、山下廣太郎氏および㈱日立製作所 システム開発本部 山中止志郎氏、花岡かほる氏、ソフトウェア開発本部 片岡雅憲氏、原田千秋氏、礒辺寛氏（現システム開発研究所）、吉村紀久雄氏、松尾洋氏、小塚潔氏、和歌山哲氏をはじめとする多くの方々に、有益なご指導、ご討論を戴くとともに問題の所在を教示頂いた。

本研究は、研究所の同僚・後輩との討論に刺激を受けて進められたものである。特に、第３章に関して、増石哲也氏、佐藤由美子氏（旧姓飯塚）、安信千津子氏、第４章に関して、安信千津子氏、山岸広基氏、第５章に関して、橋本肇氏、小堀誠氏には、共同研究者として御助言を戴き、また、プログラム開発を含め研究の一翼を担って戴いた。

上記以外にも、多くの方々のご指導とご協力を頂いた。活発な討論をして頂き、かつ、建設的な批判により著者を激励してくれた同僚・友人たちとのめぐりあいは幸運であった。

最後に、本論文の執筆にあたり終始あたたかく見守り励ましてくれた妻 節子と子供たち 泰平、亮平、修平に感謝する。

図表一覧

発表論文リスト

論文

主筆

[1] 辻 洋、増石 哲也、佐藤由美子、安信千津子：
エキスパートシステムの試行／開発／利用／保守環境，
人工知能学会誌、 Vol.4, No.4, pp.431-439 (1989)

[2] 辻 洋、安信千津子：
メタ知識定義による知識ベースの保守方式とその適用例，
人工知能学会誌、 Vol.3, No.3, pp.73-82 (1988)

[3] Hiroshi Tsuji and Hajime Hashimoto:
Expert System for Transferring Programming Knowhow From Skilled to Unskilled Programmers,
人工知能学会誌、 Vol.3, No.6, pp.755-764 (1988)

[4] Hiroshi Tsuji and Jaime G. Carbonell:
Problem Solver with Strategies for Clearing Goal Stack and Forestalling State Space, Technical Report of Carnegie Mellon University,
CMU-CMT-88-109 (1988)

[5] Hiroshi Tsuji:
Heuristic Search Strategies in MEA-Based Problem Solver and Their Case-Based Knowledge Acquisition,
人工知能学会誌、 Vol.7, No.4, pp.708-714 (1992)

[6] Hiroshi Tsuji and Fumihiko Mori:
ELISE : Office Procedure Automation Tool By State-Transition Model,
Journal of Information Processing, Vol.12, No.1, pp.9-16 (1989)

連名

[1] Tohru Katayama and Hiroshi Tsuji:
Restortion of Noisy Images by Using Two-Dimensional Linear Model,
Preprints of 4th International Joint Conference on Pattern Recognition, pp.509-511 (1978)

[2] 片山 徹、辻 洋：
２次元線形モデルによる画像の復元，
システムと制御、 Vol.23, No.12, pp.719-725 (1979)

[3] Fumihiko Mori and Hiroshi Tsuji:
A System for Decision Support without Explicit Objective Functions,
OMEGA The Int. Jl of Mgmt Sci. Vol.11, No.6, pp.567-574 (1983)

[4] Fumihiko Mori and Hiroshi Tsuji:
A Conversational Decision Support System for Resource Allocation without Explicit Objective Function,
Proc. of The AFIPS National Computer Conference, pp.1-6 (1980)

[5] Kohtaro Yamashita, Hiroshi Tsuji :
Man-Machine Conversation Control Program Generator for Computer-Based OR/MS Applications, Proc. of 14th Hawaian International Conference on System Science, pp.614-623 (1981)

[6] Fumihiko Mori and Hiroshi Tsuji :
A Behavioral Approach To Decision Support For Resource Allocation Under Uncertainty,
Proc. of Malaysian National Computer Conference, pp.235-250 (1983)

[7] Yumiko Iizuka and Hiroshi Tsuji:
A Computer System Configuration Design Expert System : IDEA/C,
Proc. of International Workshop on Artificial Intelligence for Industrial Applications (IEEE/AIIA), pp.442-447 (1988)

[8] Chizuko Yasunobu, Rei Itsuki, Hiroshi Tsuji and Fumihiko Mori :
Document Retrieval Expert System shell with Worksheet-based Knowledge Acquisition Facility,
Proc. of IEEE COMPSAC-89, pp.278-285 (1989)

解説

[1] 辻 洋、橋本 肇、山中止史郎：
ノウハウの知識ベース化によるシステム／ソフトウェアの品質管理，
（社）品質管理学会誌「品質」、 Vol.18, No.2, pp.123-129 (1988)

[2] 辻 洋、秋藤 俊介：
バージョン空間法とその応用：チューニング支援エキスパートシステム
日経インテリジェントシステム、別冊１９９２年秋号，pp.130-139 (1992)

[3] 辻 洋、飯塚由美子、山中止史郎：
計算機システム構成設計支援エキスパートシステム，
日立評論、 Vol.69, No.3, pp.279-284 (1987)

[4] 飯塚由美子、辻 洋、山中止史郎：
コンピュータのシステム構成設計を支援するエキスパートシステム，
日経エレクトロニクス、1987.2.23, pp.163-183 (1987)

講演

[1] 辻 洋、森 文彦:
目的関数が設定できない場合の対話型意志決定支援システム,
第5回システムシンポジウム講演論文集、 pp.141-146 (1979)

[2] 辻 洋、森 文彦:
対話型意志決定支援システム構築に関する2, 3の考察,
第7回システムシンポジウム講演論文集、 pp.139-144 (1981)

[3] 辻 洋、山下廣太郎、野上 昌彦:
既開発プログラムを対話型で利用するためのソフトウェア制御方式について,
第22回情報処理学会全国大会論文集、 pp.347-348 (1981)

[4] 辻 洋、山下廣太郎:
オフィス用DSSの表示機能の分析と実現方式の提案,
第24回情報処理学会全国大会論文集、 pp.1135-1136 (1982)

[5] 辻 洋、山下廣太郎、森 文彦:
オフィスにおける対話型DSSとバッチシステムの結合方式について,
第25回情報処理学会全国大会論文集、 pp.1249-1250 (1982)

[6] 辻 洋、山下 哲夫、大熊 祥訓:
非定期的なデータ更新を考慮した時系列DBMSとアプリケーションについて,
第27回情報処理学会全国大会論文集、 pp.1515-1516 (1983)

[7] 辻 洋、森 文彦:
状態遷移を考慮したオフィス手続きの蓄積法と利用法について,
第28回情報処理学会全国大会論文集、 pp.1399-1400 (1984)

[8] 辻 洋、中村 宏二、松尾 洋:
計画管理用サマリーデータのリレーショナル表現方法に関する特性比較,
第29回情報処理学会全国大会論文集、 pp.849-850 (1984)

[9] 辻 洋、森 文彦:
テーブル型データ構造をベースとしたOA用エキスパートシステム,
第34回情報処理学会全国大会論文集、 pp.1593-1594 (1987)

[10] 辻 洋、安信千津子:
知識保守のためのメタ知識表現とそのインタプリタ,
第1回人工知能学会全国大会、pp.253-256 (1987)

[11] 辻 洋、小堀 誠、橋本 肇:
ソフトウェア常識集IRシステムSOCKS (2) ー連想機能についてー,
第35回情報処理学会全国大会論文集、 pp.1507-1508 (1987)

[12] 辻 洋、安信千津子、金森 喜正、今泉 和彦
エキスパートシステム構築ツールES/X90 (5) :ー知識獲得機能. ,
第35回情報処理学会全国大会論文集、 pp.1741-1742 (1987)

その他

[1] 中村 宏二、大熊 祥訓、山下廣太郎、辻 洋:
金融機関における意志決定支援システム
日立評論、 Vol.64, No.5, pp361-364 (1982)

[2] 山下廣太郎、森 文彦、辻 洋、礒辺 寛、中村 宏二
オフィスにおける意思決定支援システム "DSS" 構築のアプローチ
実例コンピュータバンキング、No. 8, 近代セールス社刊、pp.188-195 (1982)

[3] 森 文彦、辻 洋、山下廣太郎
本部情報システム構築のアプローチ
実例コンピュータバンキング、No. 9, 近代セールス社刊、pp.111-120 (1983)

[4] 橋本 肇、神品 芳明、辻 洋、安信千津子:
ソフトウェア常識集知的検索エキスパートシステム
日立評論、 Vol.70, No.11, pp117-122 (1988)

[5] 知識処理システム専門委員会:
知識処理システムに関する調査報告書 ー知識獲得と運用ー
(社) 日本電子工業振興協会 (1989)

[6] 知識処理システム専門委員会:
知識処理システムに関する調査報告書II ー知識処理システムのブレークスルー
(社) 日本電子工業振興協会 (1990)

[7] 花岡かほる、辻 洋、丸岡哲也:
エキスパートシステム構築標準手順 "ESGUIDE"
日立評論、 Vol.72, No.11, pp1126-1130 (1990)

[8] 片山 徹、辻 洋、椹木 義一:
線形モデルによる2次元画像の同定
第9回確率システムシンポジウム、 pp.115-118 (1977)

[9] 森 文彦、辻 洋:
効用が不明確な場合に対する会話型予算配分意志決定支援システムの提案
第20回情報処理学会全国大会論文集、 pp.897-898 (1979)

[10] 山下廣太郎、辻 洋、野上 昌彦:
オンライン対話型処理化ツールの開発
第22回情報処理学会全国大会論文集、 pp.345-346 (1981)

[11] 山下廣太郎、辻 洋、森 文彦:
漢字・英数字混在データの検索表示法
第23回情報処理学会全国大会論文集、 pp.997-998 (1981)

[12] 山下廣太郎、辻 洋、森 文彦、田畑邦晃：
オフィスＤＳＳにおける表示機能について
日本語情報処理ワークショップ、ＯＡにおける日本語情報処理の活用
電子協、57-c-441，pp.180-189（1982）
[13] 飯塚由美子、辻 洋、山中止史郎：
計算機システム構成設計支援エキスパートシステムの開発
第３３回情報処理学会全国大会論文集、 pp.1409-1410（1986）
[14] 飯塚由美子、辻 洋、山中止史郎：
計算機システム構成設計支援エキスパートシステムにおける知識ベース保守機能
第３４回情報処理学会全国大会論文集、 pp.1469-1470（1987）
[15] 増石 哲也、辻 洋、安信千津子：
知識の分類基準をもつハイブリッド型知識表現
第３４回情報処理学会全国大会論文集、 pp.1665-1666（1987）
[16] 近藤 秀文、辻 洋、高志 林一、古庄 宏彰
オブジェクト指向エキスパートシステムの知識ベース管理システムにおける
知識の特徴に関する考察
第３４回情報処理学会全国大会論文集、 pp.1671-1672（1987）
[17] 近藤 秀文、辻 洋、高志 林一、花塚 光博
エキスパートシステム構築ツールＥＳ／Ｘ９０（７）：－知識ベース管理機能.
第３５回情報処理学会全国大会論文集、 pp.1745-1746（1987）
[18] 飯塚由美子、辻 洋、増石 哲也、藤波 努
エキスパートシステム構築ツールＥＳ／Ｘ９０（９）：－応用例.
第３５回情報処理学会全国大会論文集、 pp.1749-1750（1987）
[19] 橋本 肇、赤瀬 幸夫、辻 洋：
ソフトウェア常識集ＩＲシステムＳＯＣＫＳ（１）－概要－
第３５回情報処理学会全国大会論文集、 pp.1505-1506（1987）
[20] 秋藤 俊介、辻 洋：
ヒューリスティックスと動的なルール生成を用いた定性推論の効率化方式
情報処理学会第４０回全国大会論文集（1990）
[21] 藤波 努、辻 洋：
論理和を含む概念の学習アルゴリズムとその応用
情報処理学会第４０回全国大会論文集（1990）
[22] 吉浦 裕、橋本 和広、辻 洋：
Case-Based Reasoning によるエキスパートシステムの知識獲得の容易化(1)(2)
－計算機室レイアウト問題への適用－
情報処理学会第４０回全国大会論文集（1990）

[23] 秋藤 俊介、辻 洋、吉原 郁夫、高橋 広、松尾 洋、磧谷 利夫：
事例を用いたプログラムチューニング支援システム（１）（２）
情報処理学会第４１回全国大会論文集，2-59~62（1990）
[24] 藤波 努、辻 洋、広瀬 正、戸塚 健司：
ソフトウェア障害解析支援システム
情報処理学会第４１回全国大会論文集，2-23~24（1990）
[25] 花岡 かほる、辻 洋：
エキスパートシステム構築方法論ＥＳＧＵＩＤＥ
情報処理学会第４１回全国大会論文集，2-83~84（1990）
[26] 谷口 洋司、辻 洋：
表形式事例の比較・対照による分析型エキスパートシステム
情報処理学会第４２回全国大会論文集，2-213~214（1991）
[27] 秋藤 俊介、辻 洋、高橋 広、吉原 郁夫、松尾 洋、磧谷 利夫：
事例を用いたプログラムチューニング支援システム（３）
情報処理学会第４２回全国大会論文集，2-255~256（1991）
[28] 安信千津子、辻 洋、山田 弘、花岡 かほる、吉野 克之
エキスパートシステム構築標準手順：ＥＳＧＵＩＤＥとその適用
第５回人工知能学会全国大会、pp.109-112（1991）
[29] 秋藤 俊介、辻 洋：
バージョンスペースを用いた類似性の判定方法とその応用
第５回人工知能学会全国大会、pp.169-172（1991）
[30] 秋藤 俊介、辻 洋、阿部 郁男、高橋 広、吉原 郁夫、松尾 洋：
事例を用いたプログラムチューニング支援システム（４）（５）
情報処理学会第４４回全国大会論文集、pp.2-53~56（1992）
[31] 木山 忠弘、辻 洋、絹川 博之：
自然語インタフェースにおける対話型解釈内容変更方式の開発
情報処理学会第４５回全国大会論文集、pp.3-133~134（1992）
[32] 間瀬 久雄、木山 忠弘、辻 洋、絹川 博之：
自然語インタフェースにおける解釈結果確認文生成方式の開発
情報処理学会第４５回全国大会論文集、pp.3-135~136（1992）
[33] 難波 康晴、辻 洋、絹川 博之：
自然語インタフェースにおける操作対象と操作条件の表現
情報処理学会第４５回全国大会論文集、pp.3-137~138（1992）
[34] 難波 康晴、辻 洋、絹川 博之：
次世代自然語インタフェース技術－汎用意味解析処理
日本ソフトウェア科学会、ｿﾌﾄｳｪｱ研究会（関西）、SW-92-11-3、pp.15~22（1992）

参考文献

[Akifuji, et al. 1990a] 秋藤俊介、辻 洋 ：ヒューリスティックスと動的なルール生成を用いた定性推論の効率化方式，情報処理学会第４０回全国大会論文集，1990.

[Akifuji, et al. 1990b] 秋藤俊介、辻 洋、吉原 郁夫、高橋 広、松尾 洋、磧谷利夫 ：事例を用いたプログラムチューニング支援システム（１）（２），情報処理学会第４１回全国大会論文集，2-59~62，1990.

[Akifuji, et al. 1991] 秋藤 俊介、辻 洋、高橋 広、吉原 郁夫、松尾 洋、磧谷利夫，事例を用いたプログラムチューニング支援システム（３），情報処理学会第４２回全国大会論文集，2-255~256，1991.

[Anderson. 1983] Anderson, J. R. : Acquisition of Proof Skills in Geometry, Machine Learning, Vol. 1, R. S, Michalski, et al (Ed.), Morgan Kaufmann, 1983.

[Apte, et al. 1986] Apte, C. and S.J. Hong : Using Qualitative Reasoning To Understand Financial Arithmetic. Proceedings of AAAI-86, pp.942-948, 1986.

[Arai, et al. 1987] 新井政彦、本位田真一：診断型エキスパートシステム、情報処理、Vol.28, No.2, pp.177-186, 1987.

[Arisawa. 1986] 有沢 ：ソフトウェアプロトタイピグ、近代科学社、1986.

[Barstow, et al. 1983] Barstow, D. R., N. Aiello, R. O. Duda, L. D. Erman, C. L. Forgy, et al : Language and Tools for Knowledge Engineering, In F. Hayes-Rothe, et al, (Ed.) Building Expert Systems, Addison-Wesley, 1983.

[Barstow, et al. 1984] Barstow, D. R., et al : Interactive Programming Environments, McGRAW-HILL, 1984.

[Bradtke, et al. 1988] Bradtke, S. and W.G. Lehnert : Some Experiments With Case-based Search, Proc. of Case-based Reasoning Workshop, pp.80-93, 1988.

[Boem. 1976] Boem, B. W. : Software Engineering, IEEE Transaction on Computer, Vol.25, No.12, pp.1226-1241, 1976.

[Boose. 1984] Boose, J. H. : Personal Construct Theory and the Transfer of Human Expertise, Proceedings of AAAI-84, pp.27-33, 1984.

[Boose. 1985] Boose, J.H. : A Knowledge Acquisition Program for Expert Systems Based on Personal Construct Psychology, International Journal of Man-machine Studies 23, 1985.

[Boose. 1990] Boose, J. H. : Knowledge Acquisition Tools, Methods, and Mediating Representation, Proceedings of the First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop, pp.25-62, 1990.

[Buchanan, et al. 1983] Buchanan, B.G.,D. Barstow, R. Bechtal, J. Bennett, W. Clancey, C. Kulikowski, T. Mitchell, and D. Waterman : Constructing an Expert System, In F. Hayes-Rothe, et al, (Ed.) Building Expert Systems, Addison-Wesley, 1983.

[Bylander, et al. 1986] Bylander, T. and S. Mittal : CSRL - A Language for Classification Problem and Uncertainty Handling, AI Magazine, August, 1986.

[Carbonell. 1983] Carbonell, J. G. : Learning by Analogy : Formulating and Generalizing Plans from Past Experience, Machine Learning, Vol. 1, R. S. Michalski, et al (Ed.) Morgan Kaufmann, 1983.

[Carbonell, et al. 1987] Carbonell, J, G. and Y. Gil : Learning by Experimentation, Proceedings of the Fourth International Workshop on Machine Learning, Irvine, 1987.

[Chandrasekaran. 1985] Chandrasekaran, B. : Generic Task in Knowledge-based Reasoning : Characterizing and Designing Expert Systems at the Right Level of Abstraction, The national Academy of Sciences/ Office of Naval Reserch Workshop on AI and Distibuted Problem Solving, May, 1985.

[Chusho, et al. 1987] 中所武司、他 ：エキスパートシステム構築ツールＥＳ／Ｘ９０、情報処理学会第３５回全国大会、pp.1733-1750，1987.

[Chusho, et al. 1989] 中所武司、芳賀博英、増位庄一、吉浦裕：マルチパラダイム型知識処理言語における対立概念の融合方式、人工知能学会誌、Vol.4, No.1, 1989.

[Clancey. 1984] Clancey, W. : Classification Problem Solving, Proceedings of AAAI-84, pp.49-55, 1984.

[Conklin. 1987] J. Conklin : HyperText - An Introduction and Survey, IEEE COMPUTER, Vol.20, September, 1987.

[Date. 1981] Date, C. J. : An Introduction of Database Systems, Third Edition, Addision Wesley, 1981.

[Davis, et al. 1982] Davis, R. and D. Lenat : Knowledge-based Systems in Artificial Intelligence, McGraw-Hill, 1982.

[de Kleer, et al 1985] de Kleer, J. and J.S. Brown : A Qualitative Physics Based on Confluences, In Bobrow, F.G. (Ed.) Qualitative Reasoning about Physical Systems, pp.7-83, MIT Press, 1985.

[Dejong, et al. 1986] DeJong, G. F. and R. Mooney : Explanation-Based Learning : An Alternative View, Machine Learning, Vol.1, No.2, 1986.

[Ellis, et al. 1980] Ellis, C. A. and Nutt, G. J.: Office information systems and computer sciences. ACM Computer Survey Vol.12, No.1, pp.3-36, 1980.

[Eshelman, et al. 1986] Eshelman, K., et al : MOLE : A Knowledge Acquisition Tool That Uses Its Head, Proceedings of AAAI-86, 1986.

[Feigenbaum. 1977] Feigenbaum, E. : The Art of Artificial Intelligence - Themes and Case Studies of Knowledge Engineering, Proceedings of 5th International Joint Conference on Artificial Intelligence, pp.1014-1029, 1977.

[Fikes, et al. 1971] Fikes, R. and N. Nilsson : STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, Artificial Intelligence, Vol. 2, 1971.

[Fikes, et al. 1985] Fikes, R., et al : The Role of Frame-based Representation in Reasoning, Communication of ACM Vol.28, No.9, 1985.

[Funahashi, et al. 1987] 舩橋誠壽、増位庄一：制御分野におけるエキスパートシステム、情報処理、Vol.28, No.2, pp.197-206, 1987.

[Gaines, et al. 1990] Gaines, B. and M. Shaw : Foundations of Knowledge Acquisition, Proceedings of the First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop, pp.3-24, 1990.

[Ginsberg. 1985] Ginsberg, A. , S.M. Weiss, and P. Politakis : SEEK2 : A Generalized Approach to Automatic Knowledge Base Refinement, Proceedings of International Joint Conference on Artificial Intelligence '85, pp.367-374, 1985.

[Goldberg, et al. 1983] Goldberg, and D. Robson : Smalltalk-80 : The Language and its Implementation, Reading, MA, Addison-Wesley, 1983.

[Greif. 1988] I. Greif : Computer-Supported Cooperative Work - A Book of Readings, Morgan Kaufmann Publishers, 1988.

[Hammer, et al. 1980] Hammer, M and Kunin, J. S.: Design principles of an office specificaion language, AFIPS Conference, Proceedings of Nationall Computer Conference (NCC), pp.541-547, 1980.

[Hammond. 1986] Hammond, K. J. : CHEF - A Model of Case-based Planning. Proc of AAAI-86. pp.267-271, 1986.

[Hanaoka, et al. 1990] 花岡かほる、辻 洋、丸岡哲也：エキスパートシステム構築標準手順"ＥＳＧＵＩＤＥ"，日立評論、 Vol.72 No.11, pp.1126-1130, 1990.

[Hashimoto, et al. 1987] 橋本肇、赤瀬幸夫、辻 洋:ソフトウェア常識集ＩＲシステムＳＯＣＫＳ（１）－概要－、情報処理学会第３５回全国大会、1987.

[Hashimoto, et al 1988] 橋本 肇、神品 芳明、辻 洋、安信千津子：ソフトウェア常識集知的検索エキスパートシステム，日立評論、 Vol.70 No.11, pp.117-122, 1988.

[Hayes-Roth, et al 1983] Hayes-Roth, F., D.A.Waterman, andD.B. Lenat (Eds) : Building Expert Systems, Addison-Wesley, Inc., 1983.

[Hewitt. 1986] C. Hewitt : Offices are Open Systems, ACM Transaction On Office Information Sytems, Vol.4, No.3, 1986.

[Hirai. 1987] 平井：わが国におけるエキスパートシステム開発状況、人工知能学会誌、Vol.2, No.1, 1987.

[Iizuka, et al. 1987] 飯塚由美子、辻 洋、山中止志郎：計算機システム構成設計支援エキスパートシステムにおける知識ベース保守機能、情報処理学会第３４回全国大会，1987.

[Iizuka. 1988] Yumiko Iizuka and Hiroshi Tsuji : A Computer System Configuration Design Expert System : IDEA/C, Proc. of International Workshop on Artificial Intelligence for Industrial Applications (IEEE/AIIA), pp.442-447, 1988.

[Ishii. 1989] 石井：グループウェア技術の研究動向、情報処理、Vol.30, No.12

[Isobe, et al. 1982] Isobe, H. and Yamashita, K.: EXCEED - Exective Management Decision Support System, Proceedings of APL Users Meetings, Vol.1, pp.296-324, 1982.

[Ito, et al. 1984] 伊藤 潔、田畑孝一：ソフトウェア設計におけるプロトタイピング、ｂｉｔ、Vol.15, No.6, pp.756-769, 1984.

[Kanamori, et al. 1987] 金森喜正、大小田隆、増位庄一、田野俊一、中川克則：ＥＳ／ＫＥＲＮＥＬでの知識表現法と高速推論方式、日立評論、Vol.69, No.3, 1987.

[Kawaguchi. 1989] 川口敦生、溝口理一郎、角所 収 : インタビュ-システムのためのシェルＳＩＳ，人工知能学会誌、Vol. 4, No.4, pp.411-420, 1989.

[Kobayashi. 1985] 小林重信：プロダクションシステム、情報処理、Vol.26, No.12, pp. 1487-1496, 1985.

[Kobayashi, et al. 1986] 小林、外："ソフトウェアの品質管理"特集、品質、VOL.16 -NO.1, 1986.

[Kolodner. 1985] Kolodner J. L. : A Process Model of Case-based Reasoning in Problem Solving. Proc of 9th IJCAI, pp.284-290, 1985.

[Kosy, et al. 1984] Kosy, E.W. and B.P. Wise : Self-Explanatory Financial Planning Models, Proceedings of AAAI-84, pp176-181, 1984.

[Kumagai. 1988] 熊谷正夫：実用化ツールの知識ベースエディタ、人工知能学会誌，Vol. 3, No.6, pp.38-49, 1988.

[Lefkovitz, et al. 1979] Lefkovitz, H. C. et al: A Status Report on the Activities of the CODASYL End User Facilities Committee (EUFC), SIGMOD RECORD. Vol.10, 1979.

[Martin. 1986] Martin, J. : Application Development without Programmers, PRENTICE-HALL, Inc., 1982.

[McDermott. 1982] McDermott, J. : R1 - Rule-Based Configurer of Computer Systems, Artificial Intelligence, Vol.19, pp.39-88, 1982.

[McDermott. 1986] McDermott, J. : Making Expert Systems Explicit, Proceedings of 10th IFIP World CONGRESS, 1986.

[Michalski, et al. 1983] R. S. Michalski, J.G. Carbonell and T. Mitchell, Machine Learning, Vol. 1, Morgan Kaufmann, 1983.

[Minton, et al. 1987] Minton, S. and J. G. Carbonell : Strategies for Learning Search Control rules - An Explanation-based Approach, Proceedings of 10th International Joint Conference on Artificial Intelligence , 1987.

[Minton, et al. 1988] Minton, S., et al. : PRODIGY 1.0: The Manual and Tutorial, Carnegie-Mellon University Computer Science Department, 1988.

[Minton. 1988a] Minton, S. : Quantitative Results Concerning the Utility of Explanation-Based Learning, Proc. of AAAI-88, pp.564-569, 1988

[Minton. 1988b] Minton, S. : Learning Search Control Knowledge - An Explanation -Based Approach, Kluwer Academic Publishers, 1988.

[Mitchell, et al. 1983] Mitchell, T, P. Utgoff, and S. Kedar-Cabelli : Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics, In Machine Learning, Vol. 1, R. S. Michalski, et al (Ed.) Morgan Kaufmann, 1983.

[Mitchell, et al. 1986] Mitchell, T., K.M. Keller, and S.T. Kedar-Cabelli : Explanation-Based Generalization : A Unifying View, Machine Learning, Vol.1, No. 1, pp47-80, 1986.

[Mizoguchi, et a. 1988] 溝口理一郎、角所 収：知識獲得支援システム、人工知能学会誌, Vol.3, No.6, pp.50-58, 1988

[Mori, et al. 1988] Mori, F., H. Ohata, and K. Nakamura : Expert Systems in Banking, HITACHI REVIEW, Vol.37, No.5, pp.329-332, 1988.

[Mori, et al. 1988] 森 文彦、増位 庄一 ：ビジネスマンのためのＡＩ入門、オーム社、1988.

[Mori, et al. 1990] 森 文彦、花岡 かほる ：エキスパートシステム構築技法、オーム社、1990.

[Nagasawa. 1987] 長澤 勲：設計エキスパートシステム、情報処理、Vol.28, No.2, pp. 187-196, 1987.

[Nara, et al. 1984] 奈良隆正、吉田保生：ソフトウエアの品質保証、電子通信学会技術研究報告、[35], pp.7-17, 1984.

[Newell, et al. 1972] Newell, A., and H. A. Simon : Human Problem Solving. Englewood Cliffs, N. J., Prentice-Hall, 1972.

[Nilsson. 1980] Nilsson, N : Principles of Artificial Intelligence, Tioga Publishing Co., 1980.

[Niwa. 1986] Niwa, K. : A Knowledge-Based Human Computer Cooperative System for Ill-Structured Management Domains, IEEE transaction on System, Man and Cybernetics, Vol.16, No.3, pp.335-342, 1986.

[Ohba, et al. 1987] 大場みち子、都島 功：知識工学を応用したプロジェクト計画方式、情報処理学会第３５回全国大会、pp.1131-1132, 1987.

[Ogawa. 1985] 小川 均: フレーム理論に基づく知識表現言語、情報処理、Vol.26-No.12, pp.1497-1503, 1985.

[Ohkubo. 1988] 大久保、石井：オフィスプロシジャ修正事例に基づくプランニング技法、人工知能学会研究会資料、ＳＩＧ－ＨＩＣＧ－８８０１－４, 1988.

[Quinlan. 1983] Quinlan, J. R. : Learning Efficient Classification Procedures and Their Application to Chess End Games, Machine Learning, Vol. 1, R. S, Michalski, et al (Ed.), Morgan Kaufmann, 1983.

[Ruby, et al. 1989] Ruby, D. and D. Kibler : Learning Subgoal Sequences for Planning, Proc. of 11th IJCAI, pp.609-613, 1989.

[Sasaki, et al. 1986] 佐々木浩二、他：知識工学の産業への応用、人工知能学会誌、Vol.1, No.1, pp.64-71, 1986.

[Schoen, et al. 1987] Schoen, S., and W. Sykes : Putting Artificial Intelligence to Work, John Wiley & Sons, Inc., 1987.

[Shank. 1982] Shank, R. C. : Dynamic Memory. Cambridge University Press, 1982.

[Shimauchi. 1972] 島内剛一、プログラム言語論、電子計算機基礎講座５、共立出版、１９７２.

[Shu. 1985] Shu, N. C.: FORMAL: A Form-Oriented Visual Directed Application Development System, IEEE COMPUTER, August, pp.38-49, 1985.

[Shu, et al. 1982] N. C. Shu, et al : Specification of Forms Processing and Business Procedures for Office Automation, IEEE Transaction on Software Engineering, Vol.8, No.5, pp.499-512, 1982.

[Sirbu, et al. 1982] Sirbu, M., Schoichet, S., Kunin, J. S. Hammer, M. and Sutherland, J.: OAM: An Office Analysis Methodology, AFIPS Office Automation Conference, pp.317-330, 1982.

[Smith, et al. 1981] R. G. Smith and R. Davis : Frameworks for Cooperation in Distributed Problem Solving, IEEE Transactions on Systems, Man and Cybernetics, Vol.11-1, 1981.

[Suchman. 1983] L. A. Suchman : Office Procedure as Practical Action : Models of Work and System Design, ACM TOOIS, Vol. 1, No. 4, 1983.

[Suchman. 1989] L. A. Suchman : What are Models for and Do We want them to Support Automation, Proceeding of 11th World Congress IFIP, 1989

[Suzuki. 1985] 鈴木則久：オブジェクト指向、共立出版、1985.

[Tamai. 1987] 玉井哲雄：ソフトウェア開発への知識工学の応用、情報処理、Vol.28, No. 7, pp.898-905, 1987.

[Taniguchi, et al. 1991] 谷口 洋司、辻 洋，表形式事例の比較・対照による分析型エキスパートシステム，情報処理学会第４２回全国大会論文集，2-213~214，1991.

[Tsichritzis. 1982] Tsichritzis, D. C.: Form Management, Communication of ACM, Vol.25, No.7, pp.453-478, 1982.

[Tsuji, et al. 1984] 辻 洋、森 文彦：状態遷移を考慮したオフィス手続きの蓄積法と利用法について，第２８回情報処理学会全国大会論文集，pp.1399-1400，1984.

[Tsuji, et al. 1987a] 辻 洋、森 文彦：テーブル型データ構造をベースとしたＯＡ用エキスパートシステム，第３４回情報処理学会全国大会論文集，pp.1593-1594，1987.

[Tsuji, et al. 1987b] 辻 洋、安信千津子：知識保守のためのメタ知識表現とそのインタプリタ、人工知能学会第一回全国大会論文集、PP.253-256，1987.

[Tsuji, et al. 1987c] 辻 洋、小堀 誠、橋本 肇:ソフトウェア常識集ＩＲシステムＳＯＣＫＳ（２）－連想検索方式について－、情報処理学会第３５回全国大会，1987.

[Tsuji, et al. 1987d] 辻 洋、飯塚由美子、山中止志郎：計算機システム構成設計支援エキスパートシステム、日立評論、Vol.69，No.3，pp.279-284，1987.

[Tsuji, et al. 1988a] 辻 洋、橋本 肇、山中止史郎：ノウハウの知識ベース化によるシステム／ソフトウェアの品質管理，品質，Vol.18，No.2，pp.123-129，1988.

[Tsuji, et al. 1988b] 辻 洋、安信千津子：メタ知識定義による知識ベースの保守方式とその適用例、人工知能学会誌、Vol.3，No.3，pp.319-328，1988.

[Tsuji, et al. 1988c] Tsuji, H. and J. G. Carbonell : Problem Solver with Strategies for Clearing Goal Stack and Forestalling State Space, Carnegie-Mellon University, CMU-CMT-88-109, 1988.

[Tsuji, et al. 1988d] Hiroshi Tsuji and Hajime Hashimoto : Expert System for Transferring Programming Knowhow From Skilled to Unskilled Programmers, Journal of Japanese Society for Artificial Intelligence, Vol.3, No.6, pp.755-764, 1988.

[Tsuji, et al. 1989a] Hiroshi Tsuji and Fumihiko Mori : ELISE:Office Procedure Automation Tool By State-Transition Model, Journal of Information Processing, Vol.12, No.1, pp.9-16, 1989.

[Tsuji, et al. 1989b] 辻 洋、増石 哲也、佐藤由美子、安信千津子，エキスパートシステムの試行／開発／利用／保守環境，人工知能学会誌，Vol.4，No.4，pp.431-439，1989.

[Ueno, et al. 1988] 上野晴樹、他：パネル討論「深い知識」、人工知能学会研究会資料、SIG-KB-8804，1988.

[Ullman. 1980] Ullman, J.D. : DatabaseSystems, Computer Science Press Inc. 1980.

[van de Brug, et al. 1986] van de Brug, A., et al : The Taming of R1, IEEE Expert, Vol.1, No.3, 1986.

[Watanabe, et al. 1985] 渡辺俊典、安信千津子、山中止志郎：知識工学の計算機室機器レイアウトＣＡＤへの応用、日立評論、Vol.67，No.2，pp.967-970，1985.

[Winograd. 1988] T. Winograd : Special Issue on the Language/Action Perspective, ACM Transaction On Office Information Sytems, Vol.6, No.2, 1988.

[Wu, et al. 1986] Wu, H., H.W.Chun, A. Mimo : ISCS - A Tool Kit for Constructing Knowledge-Based System Configurators, Proceedings of AAAI-86, pp. 1015-1021, 1986.

[Yasunobu, et al. 1989] Chizuko Yasunobu, Rei Itsuki, Hiroshi Tsuji and Fumihiko Mori : Document Retrieval Expert System shell with Worksheet-based Knowledge Acquisition Facility, Proceedings of IEEE COMPSAC-89, pp.278-285, 1989.

[Yoshimura. 1988] 吉村 紀久雄、増石 哲也、 松井隆、塙 信弘 : ＥＳ／ＫＥＲＮＥＬ／Ｗの開発環境、日立評論、Vol.70，No.11，pp.14-19，1988.

[Yoshiura, et al. 1990] 吉浦 裕、橋本 和広、辻 洋 : Case-Based Reasoning によるエキスパートシステムの知識獲得の容易化(1)(2)，－計算機室レイアウト問題への適用－，情報処理学会第４０回全国大会論文集，1990.

[Zisman. 1977] Zisman, M. D. : Representation, Specificaion, and Automation of Office Procedures, Ph. D. Dessertation. Department of Decision Sciences, The Wharton School, University of Pennsylvania, 1977.

[Zisman. 1978] Zisman, M. D. : Office Automation: Revolution or Evolution?, MIT Sloan Management Review, Vol. 19, pp.1-16, 1978.

[Zloof. 1981] M. M. Zloof : QBE/OBE : A Language for Office and Business Automation, IEEE Computer, May 1981.

[Zloof. 1982] Zloof, M. M. : Office-by Example: A Business Language that Unifies Data and Word Processing and Electronic Mail, IBM SYSTEM Journal, Vol. 21, No.3, pp.272-304, 1982.

付　録

付録１　第５章で用いたＥＳ／ＫＥＲＮＥＬの知識表現

ここでは、第５章で述べたＳＯＣＫＳと呼ぶシステムが利用したエキスパートシステム構築ツールＥＳ／ＫＥＲＮＥＬの知識表現の概要（本論で利用している範囲を中心）について示す。詳細は構文および推論アルゴリズムについては、日立製作所ソフトウェアマニュアル（２０５０－３－６２４）ＥＳ／ＫＥＲＮＥＬ／Ｗ文法を参照されたい。

付１．１　フレーム構文

＜フレーム＞ === (<フレーム名> <特殊スロット>
　　[<スロット定義>。。。]
　　[<メソッド定義>])

・エキスパートシステムの対象とするデータをフレームを単位として構造化する。

＜特殊スロット＞ === <ｸﾗｽﾌﾚｰﾑ定義> | <ｲﾝｽﾀﾝｽﾌﾚｰﾑ定義>

・全てのフレームは、クラスかインスタンスかのいずれかであり、クラスフレームをもとに、インスタンスフレームを動的に生成することが可能である。

<ｸﾗｽﾌﾚｰﾑ定義> === super_class (<上位フレーム名> | system)
<ｲﾝｽﾀﾝｽﾌﾚｰﾑ定義> === class <上位フレーム名>

・アプリケーションとして上位フレームをもたない場合、system と定義する。そのため、全てのフレームは上位フレームをもつ。上位フレームのスロットやメソッドは、下位フレームに定義されていない場合、それが下位に継承される。

＜スロット定義＞ === <スロット名> [<スロット値>]
　　[<ファセット指定>]

・スロット値は、下に定義するメソッドにより更新される。また、ルールの条件節により、参照される。

<ファセット指定> === (<初期値ファセット> | <データタイプファセット>
　　| <値域ファセット> | <デーモンファセット>)

・タイプは実数型、整数型、文字型などを定義し、デーモンはスロット値が参照されたとき、更新されたときに起動されるプログラム名を定義する。

＜メソッド定義＞ === #method <C関数>。。。

・スロットの値の参照・更新はここで定義されるメソッドで行われる。

付１．２　ルール構文

＜ルール定義＞ === <ルール名> <ルール条件部>
　　<ルール実行部>

＜ルール条件部＞ === if <フレーム条件節>。。。

＜ルール実行部＞ === then （<メッセージ送信> | <システム処理関数発行>）。。。

・システム処理関数とはフレームの生成など知識ベース全体に対する操作を行う組込関数である。

<フレーム条件節> === (<ﾌﾚｰﾑ名> | <変数>)
　　<ｽﾛｯﾄ条件> {(and | or | かつ | または)} 。。。

・処理系は、本条件を満たすフレームの有無を検索する。

＜スロット条件＞ === の　@<スロット名> （は | が | を）
　　(<変数> とし
　　| <値表現> <比較語>)

・比較語には「である」「でない」「より大きい」「以上である」「より小さい」「以下である」の６種類がある。

付１．３　推論様式

ＥＳ／ＫＥＲＮＥＬの推論形式は、ルール条件部を満たすフレームを検索し、それがあった場合に、実行部に定義されたメッセージを送信する。一般にはこのメッセージ送信により新たにフレームの生成がなされたり、スロット値の更新がなされるので、条件を満たすフレームは動的に変化する。

（１）全てのルールに関して、条件を満たすフレームが皆無になった時点で推論を中止する。

（２）複数のフレームが条件を満たす場合、以下のいずれかの方法で競合解消を行う。

・フレームの生成あるいはそのスロット値の更新がより最新のものを優先する。
・ルールに記述されている条件節の多いものを優先する。
・入力された順序の古いものを優先する。

これらの戦略はメタルールと呼ぶ知識で記述するが本論では直接関係しないので説明は省略する。

付録２　第６章で用いたＰＲＯＤＩＧＹの知識表現

ここでは、第６章で述べたＰＲＯＤＩＧＹと呼ぶプロブレムソルバの知識表現の概要（本論で利用している範囲を中心）について示す。詳細な構文および推論アルゴリズムについては、[Minton, et al. 1988] を参照されたい。

付２．１　オペレータの構文

<オペレータ> === (<オペレータ名> (params <引数>。。。)
(preconds <オペレータ条件部>)
(effects <オペレータ実行部>)
)

・オペレータ条件部でオペレータを作用する条件となる世界（述語で表される）を記述する。オペレータ実行部でその世界の変化(述語の追加・削除)を記述する。引数は特に述語の中の変数でトレース中に明示したいものがある場合に定義する。

<オペレータ条件部> === ((and|or) <述語>。。。) | not <述語>)。。。

・add は論理積、or は論理和、not は否定を表す。

<オペレータ実行部> === ((add|del) 述語) 。。。

・オペレータ実行により、述語で表現される世界の状況が変わるため、条件を満たすオペレータは動的に変化する。状況の変化は、add による述語の追加、del による述語の削除の組み合わせにより実現する。

付２．２　制御ルールの構文

<制御ルール> === (<制御ルール名> (lhs <制御ルール条件部>)
(rhs <制御ルール実行部>)
)

・推論の決定フェーズ（６．２．２に記述）において複数の代替案が存在（lhs で記述）するとき、それらの中から案の選択、選好、拒絶を決定(rhs で記述)する。

・制御ルールが特に定義されていない場合、ゴール、オペレータなどの選択はシステムが定義順序などから自動的に決定する。

<制御ルール条件部> === ((and|or) <メタ述語>。。。) | not <メタ述語>)。。。

< メ タ 述 語 > === (current-node <node>)
| (current-goal <node> <ゴール述語>)
| (current-op <node> <オペレータ名>)
| (candidate-node <node>)
| (candidate-goal <node> <ゴール述語>)
| (candidate-op <node> <オペレータ名>)
| (known <node> 述語)

・メタ述語とは、ソルバが動作するときにその内部状態を参照する述語である。known 述語により、ある時点での世界の状況を参照する。

<制御ルール実行部> ===
((select|prefer|reject) (GOAL 述語|OPERETOR <オペレータ名>))。。。

・selectの場合、他の候補の探索は一切行われない。reject の場合に他の候補のいずれが選択されるかは別の制御ルールあるいはシステムにより決定される。prefer の場合は、探索される優先度が上げられ、枝刈りの対象とはならない。

付２．３　問題の構文

<問題記述> === <目標状態> <初期状態>

<目標状態> === (load-goal <述語>)

<初期状態> === (load-start-state <述語>)

付２．４　推論様式

状態を初期状態から目標状態に遷移するオペレータ列を求める。そのアルゴリズムは Means-Ends Analysis と呼ばれるものであり、サブゴール（目標状態の一つ前のもの）を順次展開していくものである。アルゴリズムについては、本文６．２．２で述べている。

付録３　本論で用いた構文仕様記述則

本論第４章および付録１、２で示した構文仕様はＡＮ記法[Shimauchi. 1972]に従っている。本論で用いている意味は次の通りである。

=== 左辺の表現が右辺の表現により定義されることを示す。

< > これで囲まれる文字が構文の構成要素であることを示す。

( ) |や。。。と併用するときに構成要素のグループ化を表す。

| この前後の表現のいづれかが記述されるという代替案を表現する。

[ ] 省略可能であることを示す。

... この左の要素が１回以上繰返し定義されることを示す。

{ } ２回以上繰り返す場合、それらの間に挿入するものを示す。

___ アンダライン上の文字は固定文字列である。